PhD Dissertation

International Doctorate School in Information and Communication Technologies

DIT - University of Trento

Concept Search:
Semantics Enabled Information Retrieval

Uladzimir Kharkevich

Advisor:

Prof. Fausto Giunchiglia

Università degli Studi di Trento

March 2010

Abstract

The goal of information retrieval (IR) is to map a natural language query, which specifies the user information needs, to a set of objects in a given collection, which meet these needs. Historically, there have been two major approaches to IR that we call syntactic IR and semantic IR. In syntactic IR, search engines use words or multi-word phrases that occur in document and query representations. The search procedure, used by these search engines, is principally based on the syntactic matching of document and query representations. The precision and recall achieved by these search engines might be negatively affected by the problems of (i) polysemy, (ii) synonymy, (iii) complex concepts, and (iv) related concepts. Semantic IR is based on fetching document and query representations through a semantic analysis of their contents using natural language processing techniques and then retrieving documents by matching these semantic representations. Semantic IR approaches are developed to improve the quality of syntactic approaches but, in practice, results of semantic IR are often inferior to that of syntactic one. In this thesis, we propose a novel approach to IR which extends syntactic IR with semantics, thus addressing the problem of low precision and low recall of syntactic IR. The main idea is to keep the same machinery which has made syntactic IR so successful, but to modify it so that, whenever possible (and useful), syntactic IR is substituted by semantic IR, thus improving the system performance. As instances of the general approach, we describe the semantics enabled approaches to: (i) document retrieval, (ii) document classification, and (iii) peer-to-peer search.

Keywords

Semantic Information Retrieval, Concept Indexing, P2P Search, Semantic DHT, Semantic Flooding.

Acknowledgments

I would like to thank my scientific advisor, Prof. Fausto Giunchiglia, for believing in me when no one else would and for giving me the 'one chance' to experience the extremely interesting life of a PhD student in Trento University. Also, I am thankful to Ilya Zaihrayeu who together with Prof. Giunchiglia spent countless number of hours teaching me how to do research and write articles.

I also thank all my friends who helped me to remember that the life is not only about doing the research. In particular, I am extremely grateful to two people who really become my family in Trento: my 'brother' Mikalai Krapivin and my 'sister' Volha Kerhet. These two people made really a great effort to keep my head above the water.

Finally, I thank all my real family: my mother Valentina Kharkevich, my father Mikhail Kharkevich, my sister Elena Branovec, my nephew Egorka and nice Nastenka for love, the tremendous support and inspiration which they gave me in all these years.

Contributions

This thesis makes the following contributions:

- A survey of state of the art information retrieval approaches and semantic approaches which are used in information retrieval;
- Design and development of a new approach to semantics enabled information retrieval, called concept search;
- Design and development of the algorithms which are based on the general approach and applied to the problems of document retrieval, document classification, and peer-to-peer search;
- Implementation of *semantic matching* of complex concepts, i.e., the core building block in the concept search approach, by using the inverted index technology;
- Empirical evaluation of the developed algorithms on various data sets.

Publications

The material presented in the thesis has been developed in collaboration with Fausto Giunchiglia, Ilya Zaihrayeu, Alethia Hume, Sheak Rashed Haider Noori, Dharanipragada Janakiram, Harisankar Haridas and Piyatat Chatvorawit and published in various workshops and conferences:

- [39] Fausto Giunchiglia, Ilya Zaihrayeu, and Uladzimir Kharkevich. Formalizing the get-specific document classification algorithm. In *Proceedings of ECDL*, pages 26–37, 2007.
- [34] Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu. Concept search: Semantics enabled syntactic search. In *Proceedings of SemSearch2008 workshop at ESWC*, 2008.
- [35] Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu. Concept search. In *Proceedings of ESWC*, pages 429–444, 2009.
- [33] Fausto Giunchiglia, Uladzimir Kharkevich, and Sheak Rashed Haider Noori. P2P Concept Search: Some preliminary results. In *Proceedings of SemSearch2009 workshop at WWW*, 2009.
- [47] Uladzimir Kharkevich. Automatic generation of a large scale semantic search evaluation data-set. In *Proceedings of ICSD*, 2009.
- [32] Fausto Giunchiglia, Uladzimir Kharkevich, Alethia Hume, and Piyatat Chatvorawit. Semantic Flooding: Search over semantic links. In *Proceedings of DeSWeb 2010 workshop at ICDE*, 2010.

Contents

List of Tables

List of Figures

Introduction

If the only tool you have is a hammer,
you tend to see every problem as a nail

Abraham Maslow

The goal of information retrieval (IR) is to map a natural language query, which specifies the user information needs, to a set of objects in a given collection, which meet these needs. Historically, there have been two major approaches to IR that we call syntactic IR and semantic IR. In syntactic IR, search engines use words or multi-word phrases that occur in document and query representations. The search procedure, used by these search engines, is principally based on the syntactic matching of document and query representations. The precision and recall achieved by these search engines might be negatively affected by the problems of (i) polysemy, (ii) synonymy, (iii) complex concepts, and (iv) related concepts. Semantic IR is based on fetching document and query representations through a semantic analysis of their contents using natural language processing techniques and then retrieving documents by matching these semantic representations. Semantic IR approaches are developed to improve the quality of syntactic approaches but, in practice, results of semantic IR are often inferior to that of syntactic one. In fact, most of the state of the art search engines are based on syntactic IR. There are many reasons for this, where one of them is that techniques based on semantics, to be used properly, need a lot of background knowledge which, in general, is not available. Moreover,

the current state of the art techniques in the word sense disambiguation (i.e. converting words to meanings), does not allow to achieve the high quality of output in the concept extraction process. This leads to mistakes during the query-document matching process and, consequently, to the low quality of obtained results.

The main goal of this thesis is to develop semantics-enabled IR algorithms which can work better than (or at least as good as) syntactic IR analogues in the situation when the limited knowledge and imperfect mechanism for converting natural language to formal language are used. To achieve the goal, in this thesis, we propose a novel approach to IR which extends syntactic IR with semantics, thus addressing the problem of low precision and low recall of syntactic IR. The main idea is to keep the same machinery which has made syntactic IR so successful, but to modify it so that, whenever useful, syntactic IR is substituted by semantic IR, thus improving the system performance. This is why we call this approach a semantics enabled syntactic search. Semantics can be enabled along different dimensions, on different levels, and to different extents forming a space of approaches lying between purely syntactic search and fully semantic search. We call this space the semantic continuum. In principle, a semantics enabled approach can work on the continuum from purely syntactic search to purely semantic search, performing at least as well as syntactic search and improving over it by taking advantage of semantics when and where possible. As a special case, when no semantic information is available, the semantics enabled search reduces to syntactic search, i.e., results produced by semantic and syntactic approaches are the same. The semantics enabled approaches scale as much as syntactic search can scale because semantics is seamlessly integrated in the syntactic search technology.

As an instance of the general semantics enabled approach, in Chapter 4 of this thesis, we describe a free text document retrieval approach which

we call Concept Search (*C-Search* in short). To solve the problems related to the ambiguity of natural language, namely, the problems of polysemy and synonymy, in *C-Search*, we move from words, expressed in a natural language, to concepts (word senses), expressed in an unambiguous formal language. To solve the problem related to complex concepts, we analyze natural language phrases, which denote these concepts. The problem with related concepts is addressed by incorporating lexical knowledge about term relatedness. *C-Search* is based on the semantic matching of complex concepts, where semantic matching is implemented by using (positional) inverted index.

Using search engines is not the only way to discover the relevant information. Classification hierarchy is another major approach for improving the information discovery. It have always been a natural and effective way for humans to organize their knowledge about the world in such a way, that a person, who navigates the classification, will be facilitated in finding objects related to a given topic. These hierarchies are rooted trees where each node defines a topic category. Child nodes' categories define aspects or facets of the parent node's category, thus creating a multifaceted description of the objects which can be classified in these categories. To attain such organization of objects, in standard classification approaches, objects are manually classified by human classifiers which follow a predefined system of rules. The actual system of rules may differ widely in different classification approaches, but there are some generic principles which are commonly followed. These principles make the ground of the get-specific algorithm, which requires that an object is classified in a category (or in a set of categories), which most specifically describes the object. In Chapter 5 of this thesis, we present a first attempt to formalize the get-specific document classification algorithm and to fully automate it through reasoning in a propositional concept language without requiring a user in-

volvement or a training data-set. The get-specific algorithm is an example of a semantics enabled document classification algorithm.

The current web is a huge repository of documents, distributed in a network of autonomous information sources (peers). The number of these documents keeps growing significantly from year to year making it increasingly difficult to locate relevant documents while searching on the web. In addition to the massiveness, the web is also a highly dynamic system. Peers are continually joining and leaving the network, new documents are created on peers, and existing ones are changing their content. The search problem becomes even more complex. Nowadays, the major search engines are based on a centralized architecture. They attempt to create a single index for the whole Web. But the size, dynamics, and distributed nature of the Web make the search problem extremely hard, i.e., a very powerful server farm is required to have complete and up-to-date knowledge about the whole network to index it. The peer-to-peer (P2P) computing paradigm appeared as an alternative to centralized search engines for searching web content. Each peer in the P2P network organizes only a small portion of the documents in the network, while being able to access the information stored in the whole network. Robustness and scalability are the major advantages of the P2P architecture over the centralized architecture. Also, as the requirements for computational and storage resources of each peer in a P2P network are much lighter than for a server in a centralized approach, a peer's search engine can employ much more advanced techniques for search, e.g. semantic search.

In this thesis, we describe two approaches to semantics enabled P2P search: *P2P C-Search* (Chapter 6) and *Semantic Flooding* (Chapter 7). *P2P C-Search* extends *C-Search* allowing semantic search on top of distributed hash table (DHT). The key idea is to exploit distributed, rather than centralized, background knowledge and indices. Centralized document in-

dex is replaced by distributed index build on top of DHT. The reasoning with respect to a single background knowledge is extended to the reasoning with respect to the background knowledge distributed among all the peers in the network.

Semantic Flooding algorithm is based on the following idea. Links in classification hierarchies codify the fact that a node lower in the hierarchy contains documents whose contents are more specific than those one level above. In turn, multiple classification hierarchies can be interconnected by semantic links which represent mappings among them and which can be computed, e.g., by ontology matching. In Chapter 7 of this thesis, we describe how these two types of links can be used to define a semantic overlay network which can cover any number of peers and which can be flooded to perform semantic search on links, i.e., to perform semantic flooding. In our approach, only a relatively small number of peers need to be queried in order to achieve high accuracy.

Structure of the Thesis

The thesis is organized in four parts. Part I provides an overview of syntactic/semantic centralized/distributed IR approaches. Part II introduces and describes the semantics enabled IR approach. Part III extends the semantics enabled IR approach to the case of semantics enabled P2P search. Part IV provides an evaluation of the algorithms described in Parts II and III. Each part consists of several chapters, as follows:

Part I State of the Art

- Chapter 1 provides an overview of classical IR models and data structures. Syntactic matching of document and query terms is discussed and its problems are highlighted.

- Chapter 2 introduces semantic continuum and describes three dimensions where semantics can improve syntactic IR approaches.
- Chapter 3 provides an overview of existing state-of-the-art IR approaches in the P2P networks.

Part II Semantics Enabled Information Retrieval

- Chapter 4 introduces and describes the semantic enabled document retrieval algorithm.
- Chapter 5 describes semantics enabled algorithms for hierarchical document classification.

Part III Semantics Enabled P2P Search

- Chapter 6 describes an implementation of semantics enables document retrieval algorithm on top of the distributed hash table.
- Chapter 7 shows how links in classification hierarchies together with links across the classifications of different peers can be used to implement semantic flooding algorithm in P2P networks.

Part IV Evaluation

- Chapter 8 presents an approach for *automatic* generation of IR data-sets based on search engines query logs and data from human-edited web directories.
- Chapter 9 provides evaluation results for the algorithms described in Chapters 4, 5, 6, and 7.

Part I

State of the Art

Chapter 1

Information Retrieval

The goal of an IR system is to map a natural language query q (in a query set Q), which specifies a certain user information needs, to a set of documents d in the document collection D which meet these needs, and to order these documents according to their relevance to q. IR can therefore be represented as a mapping function:

$$IR : Q \rightarrow D \tag{1.1}$$

In order to implement an IR System we need to decide (i) which models (*Model*) are used for document and query representation, for computing query answers and relevance ranking, (ii) which data structures (*Data Structure*) are used for indexing document representations in a way to allow for an efficient retrieval, (iii) what is an atomic element (*Term*) in document and query representations, and (iv) which matching techniques (*Match*) are used for matching document and query terms. Thus, an IR System can be abstractly modelled as the following 4-tuple:

$$IR\_System = \langle Model,\ Data\ Structure,\ Term,\ Match \rangle \tag{1.2}$$

In the rest of this chapter, we will briefly describe the classical IR models (Section 1.1), data structures (Section 1.2), and the term matching process (Section 1.3). Readers interested in a detailed discussion of IR systems and

Queries:
Q1: *Babies and dogs* Q2: *Paw print* Q3: *Computer table* Q4: *Carnivores*
Documents:
D1: *A small baby dog runs after a huge white cat.*
D2: *A laptop computer is on a coffee table.*
D3: *A little dog or a huge cat left a paw mark on a table.*

Figure 1.1: Queries and a document collection

their components are referred to [59]. The problems which may negatively affect the performance of IR systems are discussed in Section 1.4. Section 1.5 summarizes the chapter.

1.1 Models

The *bag of words model* [59], i.e., the model in which the ordering of words in a document is not considered, is the most widely used model for document representation. The *boolean model* [59], the *vector space model* [79], and the *probabilistic model* [23] are the classical examples of models used for computing query answers and relevance ranking.

In the boolean model [59], documents are represented by using the bag of words model. Queries are represented as boolean expressions of terms, i.e., terms are combined with the operators *AND*, *OR*, and *NOT*. Document is considered to be an answer to a query if the boolean condition in the query is satisfied. For instance, a document $D2$ (in Figure 1.1) is an answer to a query *"laptop AND coffee AND (NOT dog)"* because both words *laptop* and *coffee* appears in the document and there is no word *dog*. One of the problems with boolean retrieval is that, in the large document collection, the number of documents which match the query can also be large, i.e., it can be bigger than a user is willing to sift through. In order to address this problem, conventional search engines rank query results according to their

relevance to the query. Different models for estimating the document-query relevance are discussed below.

In the vector space model [79], documents and queries are represented as vectors in a common vector space, in which there is an axis for each term. A component corresponding to a term t in a document (query) vector represents the importance of t in the document (query). Different measures (based on the statistics of term occurrences) were proposed to weight the term importance [62], where the *tf-idf* (*term frequency - inverse document frequency*) is probably the most popular one. The *tf-idf* weighting scheme assigns high weights to terms which appear frequently but within a small number of documents. Relevance score between a query and a document, in the vector space model, is computed by using the *cosine similarity* [105] between the vector representations of the query and the document. For instance, the scoring function which is derived from the vector space model and which is used in Lucene[1] (an open source IR toolkit used in many search applications[2]) is shown below[3]:

$$score_1(q,d) = \sum \frac{f^{\frac{1}{2}}(1+ln(\frac{N}{n+1}))^2}{dl^{\frac{1}{2}}} \tag{1.3}$$

where f - the frequency of occurrence of the term within the document; dl - the document length measured as the number of indexing terms in the document; n - the number of documents containing the term; N - the number of documents in the document collection.

In the probabilistic model [23], documents are ranked according to the probability of being relevant to the user information need which is uncertainly expressed by the user query. According to the *Probability Ranking Principle (PRP)* [72], if *"the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the sys-*

[1] `http://lucene.apache.org/java/docs/index.html`

[2] `http://wiki.apache.org/lucene-java/PoweredBy`

[3] Note that Formulas 1.3, 1.4, and 1.5 are simplified for the sake of presentation

tem for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data". Different ways of estimating these probabilities (or measures related to probabilities which don't change the ordering of results) have been proposed in the literature (e.g., see [94, 73, 85]). An example of the popular probabilistic scoring function *Okapi BM25* [74] is shown below:

$$score_2(q,d) = \sum ln(\frac{N-n+\frac{1}{2}}{n+\frac{1}{2}}) \frac{(k_1+1)f}{k_1((1-b)+b\frac{dl}{avdl})+f} \tag{1.4}$$

where k_1 and b (which default to 1.2 and 0.75 respectively) are the tuning parameters; $avdl$ is the average document length across the document collection.

Using of statistical language models [67, 42], which assign a probability for generating of an arbitrary sequence of terms, is an alternative probabilistic approach which is used in IR. In a basic language modelling approach, a language model is built for each document in the document collection. A relevance score between a query and a document is computed as a probability of generating the query terms by the document's language model. A language modelling approach can also be used to build the language model for the query [51]. The relevance score, in this case, is computed by estimating the probability of generating the document by the query's language model. In [50], authors show how the above two approaches can be efficiently combined. The simplest (yet the most widely used) form of language model is the unigram language model. The unigram model assigns each term a probability of occurrence independently from other terms. An example of a scoring function, derived from the unigram language model, is shown below [100]:

$$score_3(q,d) = ql \cdot ln(\frac{\mu}{dl \cdot \mu}) + \sum ln(1+\frac{f}{\mu\frac{n}{N}}) \tag{1.5}$$

where, ql is the query length and μ is the tuning parameter. In higher order

models, e.g., the bigram language model, the probabilities are assigned to terms depending on the previous terms. The use of higher order models in IR is discussed in [29].

In the discussed above models, the user can be involved in the retrieval process (and improve the results of IR) by using a *relevance feedback* (RF) [78] mechanism. After receiving an initial set of retrieval results, the user specifies which results are relevant to his information need and which are not. The query representation is then modified to take the feedback information into account and to produce a final retrieval results. This procedure can be repeated as many times as needed. The Rocchio Algorithm [75] is an example of how the relevance feedback information can be incorporated into the vector space model. Instead of manual selection of the relevant results, *top-k* ranked documents from the initial set of results can be considered relevant. This is the basic idea of the technique which is called *pseudo relevance feedback* [78].

1.2 Data Structures

Various index structures, such as the *signature file* [106] and the *inverted index* [105], are used as *data structures* for efficient retrieval. Inverted index, which stores mappings from terms to their locations in documents, is the most popular solution [59]. The two parts of an inverted index are: *Dictionary*, i.e., a list of terms used for document indexing; and posting lists (*Postings*), where every posing list is associated with a term and consists of documents in which this term occur. Note that the inverted index dictionary is kept in main memory and postings are kept on the disk. Different techniques for an efficient index construction where developed in the last years (e.g., see [105, 52, 17, 3]). For large document collections using of a single machine can be insufficient and, therefore, indexing may need

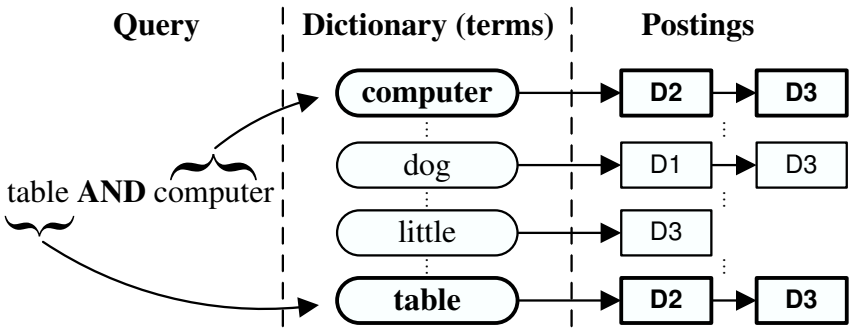


Figure 1.2: Inverted Index

to be distributed over multiple (hundreds or thousands of) machines. Application of a general MapReduce [26] distributed computing architecture is the popular solution for the distributed document indexing.

The query processing in Inverted Index is separated into three main steps: (i) to locate terms in dictionary which match query terms, (ii) to retrieve postings for these terms, and (iii) merge (e.g., intersect or unite) the postings. An example of boolean retrieval using Inverted Index technology is given in Figure 1.2. We are processing a query *table AND computer*. First, for each query term we identify those terms in dictionary that match this term ($table \rightarrow \{table\}$ and $computer \rightarrow \{computer\}$). Second, we search inverted index with computed dictionary terms ($table \rightarrow \{D_2, D_3\}$ and $computer \rightarrow \{D_2, D_3\}$). And finally, we take the intersection of document sets, found for every query terms, as an answer to the query (D_2 and D_3 in our example).

The fast search in the inverted index dictionary is usually implemented by using search trees [48]. Algorithms for an efficient access and merging of posting lists, which are based on skip lists, are discussed in [61, 13, 90]. Note that users usually see only the *top-k* retrieval results. Different heuristics which allow for a fast retrieval of the *top-k* results were proposed in [2, 30, 4].

1.3 Term Matching

In the simplest case, document and query words[4] can be considered as *terms* in document and query representations. A potential problem with this approach is that different inflectional forms of a word can be used in a query and a document and, therefore, the document will not be retrieved as an answer to the query. In order to address this problem, the inflectional forms of words can be reduced to a common base form which is used as a *term*. For instance, the stemmer can remove the ends of words by using a set of rules [68]. Furthermore, in order to reduce the number of mismatches, terms are usually converted to lower case.

Basic term matching is implemented as a search for identical terms. Furthermore, some systems perform approximate matching by searching for terms within a certain edit distance with a given term [104]. Spelling error correction and other forms of query refinement [40] can also be seen as a kind of approximate term matching. The usage of wildcard queries for term matching is covered in Chapter 3 of [59]. A wildcard query is a query which uses special characters which match any character or an arbitrary sequence of characters. For instance, a query *cat** will match documents with terms *cats* and *catastrophe*. The described above approaches are all examples of syntactic matching. Hereafter, IR systems which use syntactic matching of terms are referred to as *syntactic search* systems.

1.4 Problems

There are several problems which negatively affect the performance of syntactic search, as discussed below.

Polysemy (Figure 1.3). The same word may have multiple meanings

[4] Also phrases and, in general, any other character sequences which can not be classified as words, e.g., dates, ip addresses, emails, etc

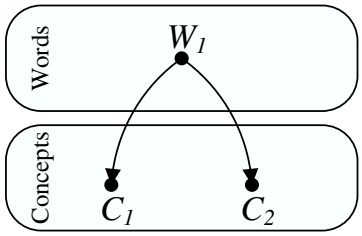


Figure 1.3: Polysemy

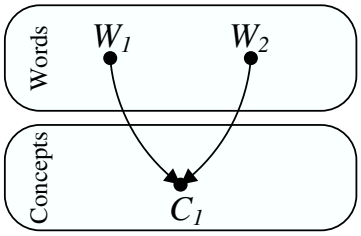


Figure 1.4: Synonymy

and, therefore, query results may contain documents where the query word is used in a meaning which is different from what the user had in mind when she was defining the query. For instance, a document $D1$ (in Figure 1.1) which talks about *baby* in the sense of a very young mammal is irrelevant if the user looks for documents about *baby* in the sense of a human child (see query $Q1$ in Figure 1.1). An answer for query $Q1$, computed by a syntactic search engine, includes document $D1$, while the correct answer is the empty set.

Synonymy (Figure 1.4). Two different words can express the same meaning in a given context, i.e., they can be synonyms. For instance, words *mark* and *print* are synonymous when used in the sense of a visible indication made on a surface, however, only documents using word *print* will be returned if the user query was exactly this word. An answer for query $Q2$ (in Figure 1.1), computed by a syntactic search engine, is the empty set, while the correct answer includes document $D3$.

Complex concepts (Figure 1.5). Syntactic search engines fall short in taking into account complex concepts formed by natural language phrases and in discriminating among them. Consider, for instance, document $D2$ (in Figure 1.1). This document describes two concepts: *a laptop computer* and *a coffee table*. Query $Q3$ (in Figure 1.1) denotes concept *computer table* which is quite different from both complex concepts described in $D2$, whereas a syntactic search engine is likely to return $D2$ in response to

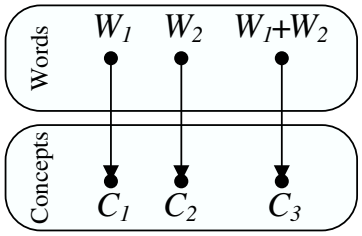


Figure 1.5: Complex concepts

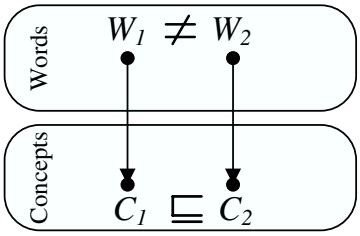


Figure 1.6: Related concepts

$Q3$, because both words computer and table occur in this document. The correct answer to $Q3$ is the empty set.

Related concepts (Figure 1.6). Syntactic search does not take into account concepts which are semantically related to the query concepts. For instance, a user looking for *carnivores* might not only be interested in documents which talk about carnivores but also in those which talk about the various kinds of carnivores such as *dogs* and *cats*. An answer for query $Q4$ (in Figure 1.1), computed by a syntactic search, is the empty set, while the correct answer might include documents $D1$ and $D3$, depending on user information needs and available semantic information.

1.5 Summary

This chapter provided a brief overview of classical IR models and data structures. Various approaches to syntactic matching of terms were also discussed. Finally we identified several problems which negatively affect the performance of syntactic IR approaches (the approaches which are based on the syntactic matching of terms).

Chapter 2

Semantic Search

Semantic search is a research topic that has its roots in the IR community that proposed first approaches to extending the classical IR with explicit semantics long time ago (e.g., see [24]). Since then many approaches were proposed by the community. However, their core common idea to codify the explicit semantics was in the use of informal knowledge representation structures such as thesauri with no or little formal reasoning support. On the other hand, with the advent of the Semantic Web (SW), many formal frameworks to represent and reason about knowledge were proposed. However, in the SW community, semantic search is primarily seen as the data retrieval task. RDF graph is queried with a query in a formal language (such as SPARQL) in order to retrieve elements of the graph satisfying the given query. See [43, 58] for an overview of approaches proposed so far in both IR and SW communities.

In the rest of this chapter, we will concentrate on the document retrieval problem where documents and queries are represented as a free text. We will see how the semantic search approaches can be used to address the problems of syntactic search described in Section 1.4. We identify three dimensions where semantics can improve syntactic search and represent these dimensions in the cartesian space shown in Figure 2.1. In Section 2.1, we

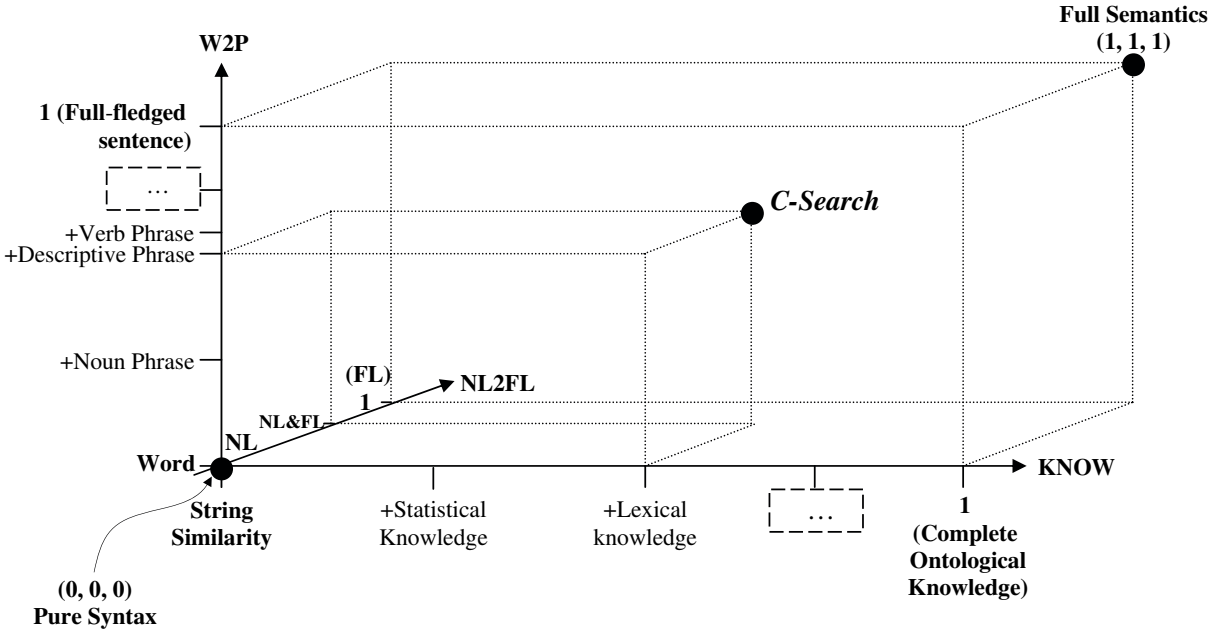


Figure 2.1: Semantic Continuum

will discuss approaches in which concepts are used for document indexing and retrieval (NL2FL-axis in Figure 2.1). In the NL2FL-axis, value 0 represents the situation where only words are used, while value 1 represents the situation where only concepts are used. In Section 2.2, we will discuss approaches where indexing by words is extended to indexing by phrases (W2P-axis in Figure 2.1). In the W2P-axis, value 0 represents the situation where only single words are used, while value 1 represents the situation where noun phrases or complex concepts, extracted from full-fledged sentences, are used. In Section 2.3, we will see how knowledge about term relatedness can be incorporated into a term matching process (KNOW-axis in Figure 2.1). In the KNOW-axis, value 0 represents the situation where only string similarity is used, while 1 represents the situation where complete ontological knowledge is used. The three-dimensional space contained in the cube (see Figure 2.1) represents the *semantic continuum* where the origin (0,0,0) is a purely syntactic search, the point with coordinates (1,1,1) is full semantic search, and all the points in between

represent search approaches in which semantics is enabled to different extents.

2.1 From Natural Language to Formal Language

The fact that the syntactic nature of classical IR (which is based on syntactic matching of ambiguous words) leads to problems with precision and recall was recognized by the IR community a long time ago (e.g., see [88]). To solve the problems related to the ambiguity of natural language, namely, the problems of polysemy and synonymy, we need to move from words, expressed in a natural language, to concepts (word senses), expressed in an unambiguous formal language. This process is commonly referred to as *Word Sense Disambiguation (WSD)* [64].

An overview of existing approaches to a sense (concept) based IR is presented in [80]. The common pattern is to use a WSD technique in order to associate words in a document corpus with atomic lexical concepts in a linguistic database and then to index these documents with the associated concepts. Different linguistic databases were used in the sense based IR, where *WordNet* [60] is a particularly popular one. The approach described in [88] is an example of the sense based IR approach which is based on *WordNet*. WSD in [88] is based on the information about words which co-occur with a word in a document, collocations which contain the given word and the frequency of senses for the word from *WordNet*. The most frequent sense is assigned to a word if there is no enough information which can be used to perform WSD. Given the lack of context in queries, which usually consist of few words, query terms in [88] are disambiguated by using the most frequent sense heuristic only.

When we move from words to concepts, it is not always possible to find

a concept which corresponds to a given word. The main reason for this problem is the lack of background knowledge [31, 37], i.e., a concept corresponding to a given word may not exist in the lexical database. To address this problem, indexing and retrieval in the continuum can be performed by using both syntactic and semantic information. For example, Hybrid Search [10] combines syntactic search with semantic search, where semantic search is implemented on metadata and syntactic search is implemented on keywords. The results of both techniques are then intersected and ranked according to scores computed by syntactic search. In [18], ontology-based search is combined with the classical IR. First, documents are scored by using each technique separately and then the final score is computed as a linear combination of scores computed by these techniques.

2.2 From Words to Phrases

To solve the problem related to complex concepts, natural language phrases (which denote these concepts) need to be analyzed. It is well known that in natural language concepts are expressed mostly as noun phrases [84]. In general, concepts can be expressed as more complex phrases than noun phrases (e.g., verb phrases) and possibly as arbitrary complex full-fledged sentences.

An example of a noun phrase parsing algorithm and its application to document indexing and retrieval is described in [99]. The parsing is performed by a fast probabilistic noun phrase parser. Indexing and retrieval are performed by using both words and phrases. It is suggested that, in this case, some parsing errors may be tolerable. Using of different combinations of words and phrases is discussed in [99]. For instance, single words can be combined with full noun phrases, head modifier pairs, or both.

There are approaches in which the conceptual content of noun phrases is

also analyzed. For instance, in [1], noun phrases are automatically translated into ontological descriptors by using a semantic analysis which is based on a domain-specific ontology. The goal of the semantic analysis is to ensure that noun phrases with similar conceptual content receive the same or similar descriptions. The retrieval is performed by matching descriptors extracted from queries with those extracted from documents.

2.3 From String Similarity to Semantic Similarity

The problem with related concepts can be solved by incorporating knowledge about term relatedness into the retrieval process. For instance, it can be statistical knowledge about term co-occurrence (e.g., see [27]), lexical knowledge about synonyms and related terms (e.g., see [63]), or ontological knowledge about classes, individuals, and their relationships (e.g., see [18]).

Latent Semantic Analysis (LSA) [27] is an example of a technique which makes use of knowledge about term co-occurrence. Singular-value decomposition (SVD) [89] and low-rank approximation [9] are used to approximate term-document matrix with a new matrix which can be seen as a linear combination of small number (e.g., 100) of factors or latent concepts. Note that in this case, a concept is a weighted vector of related words (i.e., words with similar co-occurrences). The key idea of LSA is to describe queries and documents as a linear combination of latent concepts and to perform query-document matching by comparing the latent concepts instead of words. Note that, in this case, a document can be found similar to the query even though not all the query words appear in the document. A probabilistic extension of the LSA technique is provided in [44, 12].

Lexical knowledge about synonyms and related terms is used in [95, 63, 46, 54, 97]. Roget's thesaurus [76] and WordNet [60] are examples

of the popular lexical resources which are used for this purpose. In the simplest case, terms in a query are expanded with related terms from the thesaurus and then the expanded query is used in the retrieval process (e.g., see [95, 63]). The expanded terms are usually weighted differently from the original query terms (e.g., see [46]). In [54], the combination of search technique which are based on detecting phrases and query expansion is discussed. It is argued that phrases, which are extracted from the query, should be considered more important for document retrieval than the single words. In [97], natural language processing and machine learning techniques are used to identify (noun) phrases in a document corpus, to analyze the structure and content of these phrases, and to organize them in a subsumption hierarchy. The resulting hierarchy is used to make connections between (possibly different) terminologies in the user query and indexed documents. On of the problems in using manually built thesauri is that many terms and relations can be missing, hence, affecting the performance of IR systems which use these thesauri. To overcome this problem it is suggested to replace (or to enrich) manually built thesauri by automatically generated ones. Automatic thesaurus generation and its application to IR is discussed in [69, 81, 57].

In [18], query expansion is implemented by using domain ontologies instead of thesauri. Documents are annotated by concepts and instances from an ontology. A reasoning engine is used to find concepts and instances which satisfy the query and which are in the ontology. Documents which are annotated with these concepts and instances are considered to be relevant to the query. Document ranking is implemented by adopting the cosine similarity from the vector space model where terms are replaced with atomic elements from the ontology.

2.4 Summary

In this chapter, we introduced semantic continuum and identified three dimensions in the continuum where semantics can be enabled in syntactic IR approaches. Different search approaches in which semantics is enabled to different extents and along different dimensions were discussed.

Chapter 3

P2P Search

The current web is a huge repository of documents, distributed in a network of autonomous information sources (peers). The number of these documents keeps growing significantly from year to year making it increasingly difficult to locate relevant documents while searching on the web. In addition to the massiveness, the web is also a highly dynamic system. Peers are continually joining and leaving the network, new documents are created on peers, and existing ones are changing their content. The search problem becomes even more complex.

Nowadays, the major search engines are based on a centralized architecture. They attempt to create a single index for the whole Web. But the size, dynamics, and distributed nature of the Web make the search problem extremely hard, i.e., a very powerful server farm is required to have complete and up-to-date knowledge about the whole network to index it. The peer-to-peer (P2P) computing paradigm appeared as an alternative to centralized search engines for searching web content. Each peer in the P2P network organizes only a small portion of the documents in the network, while being able to access the information stored in the whole network. Robustness and scalability are major advantages of the P2P architecture over the centralized architecture. Also, as the requirements for computational

and storage resources of each peer in a P2P network are much lighter than for a server in a centralized approach, a peer's search engine can employ much more advanced techniques for search, e.g. semantic search.

A number of P2P search approaches have been proposed in the literature (for an overview see [71]). In the rest of this chapter, we will discuss some of these approaches concentrating on syntactic search in unstructured (Section 3.1) and structured (Section 3.2) networks. In Section 3.3, approaches implementing semantic search in P2P networks will be discussed.

3.1 Search in Unstructured Networks

The algorithm implemented by Gnutella is the classical example of a query flooding algorithm. In early versions of Gnutella, connections between peers were made mainly chaotically. A P2P network was completely *unstructured*, i.e., it did not have any predefined structure. The query sent by a peer was propagated to all the actively connected peers within a predefined number of *hops* from the query sender. The search process was *blind*, i.e., peers have no information related to the resource location. The lack of scalability was recognized as the main problem of the Gnutella. Various techniques were adopted in later versions of Gnutella protocol in order to make the search process more scalable. *Super-peers* were introduced to utilize the heterogeneity between peers in computer power, bandwidth and availability. *Informed search*, i.e., when peers maintain additional information about resource locations which can be useful for the search, replaced *blind search*. In Gnutella, informed search is implemented by using Query Routing Protocol (QRP). Query Routing Tables (QRT) consisting of hashed keywords are exchanged between peers. During query routing, search request is propagated only to those peers which have all of the query words in its QRT. In [21], a peer uses Routing Indices to forward queries to

neighbors that are more likely to have answers. Query topics are compared to neighbor's expertise to select relevant peers.

The basic idea of [6, 20, 86, 103, 22] is to organize peers into Similar Content Groups on top of unstructured P2P systems, i.e., a *peer clustering* approach is implemented. Peers from the same group tend to be relevant to the same queries. A query is guided to Similar Content Group that is more likely to have answers to the given query and then the query is flooded within this group. For instance, in Semantic Overlay Networks (SONs) [22] peers that have similar documents are clustered at the same group. A predefined classification hierarchy is used to classify the peers' documents. Thus two peers belong to the same SON if some of their documents classified under the same concept in this global classification. Peers can belong to more than one SON.

3.2 Search in Structured Networks

CAN [70], Chord [87], Pastry [77], and Tapestry [102] use another approach to the routing and topology organization of P2P networks. This approach employs the idea of distributed hash tables (DHT) functionality (e.g. mapping keys onto values) on Internet-like scale. In DHT, every object is associated with a key, which is transformed into a hash using some hash function. The range of the output values of the hash function forms an ID space. Every peer in the network is responsible for storing a certain range of keys. Values, e.g., objects or information about objects, are stored at the precisely specified locations defined by the keys. A *data clustering* approach is implemented, i.e., similar data is placed in the same place. Such systems are highly *structured*. Their topology is tightly controlled. Search in these systems is limited to an exact key search. The two main operations provided by DHT are:

- *put (key, value)* - stores the *value* on the peer responsible for the given *key*.

- *get (key)* $\rightarrow$ *value* - finds a peer responsible for the *key* and retrieve the *value* for the *key*.

A straightforward way to implement syntactic search is to use the DHT to distribute peers' inverted indices in the P2P network [71]. Peers locally compute posting lists $P(t)$ for every term t and store them in the network by using the DHT 'put' operation. The *key* in this case is a term t while the *value* is a posting list $P(t)$ associated with t. In DHT, each peer is responsible for a few terms and for every term t the peer merges all the posting lists $P(t)$ for t from all the peers in the network. In order to find a set of documents which contain a term t we just need to contact the peer responsible for t and retrieve the corresponding posting list. The DHT 'get' operation does exactly this. In order to search for more than one term, we, first, need to retrieve posting lists for every single term, and then to intersect all these posting lists.

The above approach has several problems (see e.g. [53, 91]). Let us consider some of these problems.

Storage. For a large document collection, the number and the size of posting lists can be also large. Therefore, the storage needed to store the posting lists can potentially be bigger than the storage peers can (or want to) allocate.

Traffic. Posting lists need to be transferred when peers join or leave the network. Searching with multiple terms requires intersection of posting lists, which also need to be transferred. In the case of huge posting lists, a bandwidth consumption can exceed the maximum allowed. In [53], it is shown that the efficiency of DHT can be even worse than the efficiency of a simple flooding algorithm.

Load balancing. Popularity of terms, i.e., the number of occurrences of the terms, can vary enormously among different terms. It can result in an extremely imbalanced load e.g., some peers will store and transfer much more data than others.

Several approaches were proposed in order to address the described above problems and to improve performance of IR in structured P2P networks. Some of the optimization techniques (e.g., Bloom Filters), which can improve the performance of posting lists intersection, are summarized in [53]. Caching of results for queries with multiple terms is discussed in [11, 83]. In [83], only those queries are cached which are frequent enough and simple flooding is used for rare queries. In [91], only important (or top) terms are used for indexing of each document. Moreover, the term lists are stored on peers responsible for these top terms. Notice that by using only the top terms we can decrease the quality of search results. Automatic query expansion is proposed as a way to address this problem [91]. Some techniques to balance the load across the peers are also presented in [91]. Normally users are interested only in a few (k) high quality answers. An example of the approach for retrieving *top k* results, which does not require transmitting of entire posting lists, is discussed in [101]. In [55], indexing is performed by terms and term sets appearing in a limited number of documents. Different filtering techniques are used in [55] in order to make vocabulary to grow linearly with respect to the document collection size. In [8, 7], it was proposed to index a peer containing a document and not the document itself. At search time, first, those peers are selected, which are indexed by all the terms in the query, then, the most promising peers are selected, and finally, local search is performed on these peers.

Ambiguity. Another important problem with the above approaches is that all of them implement syntactic search. Therefore, the problems of syntactic search, i.e., problems of (i) polysemy, (ii) synonymy, (iii) complex

concepts, and (iv) related concepts (see Section 1.4), can also affect the quality of the results produced by these approaches.

3.3 Semantic P2P Search

All of the described so far approaches are based on syntactic matching of terms. In this section, we will discuss P2P search approaches which use matching techniques which use the knowledge about term relatedness (and not only syntactic similarity of terms). For instance, statistical knowledge about term co-occurrence is used in [92]. Knowledge about synonyms and related terms is used in [56].

Different semantic search approaches are also used in Edutella [65] and Bibster [41]. These two approaches are built on JXTA framework and aim to combine meta-data with P2P networks. Each peer is described and published using an advertisement, which is an XML document describing a network resource. For example in the Bibster [41] system, these expertise descriptions contain a set of topics that the peer is an expert in. Peers use a shared ontology to advertise their expertise in the Peer-to-Peer network.

3.4 Summary

In this chapter, we discussed various state-of-the-art IR approaches to syntactic and semantic search in structured and unstructured P2P networks.

Part II

Semantics Enabled Information Retrieval

Chapter 4

Concept Search

C-Search is a document retrieval approach which is implemented according to the model described in Equations 1.1 and 1.2 from Section 1.4. In our proposed solution, *C-Search* reuses retrieval models (*Model*) and data structures (*Data Structure*) of syntactic search with the only difference in that now words (W) are substituted with concepts (C) and syntactic matching of words ($WMatch$) is extended to semantic matching of concepts ($SMatch$). This idea is schematically represented in the equation below:

$$\boxed{\textit{Syntatic Search} \xrightarrow{Term(W \rightarrow C),\ Match(WMatch \rightarrow SMatch)} \textit{C-Search}}$$

Material presented in this chapter has been developed in collaboration with Fausto Giunchiglia and Ilya Zaihrayeu and published in [34, 35].

In the rest of this chapter, we will consider in detail how the words in W are converted into the complex concepts in C (Section 4.1) and also how the semantic matching $SMatch$ is implemented (Section 4.2). Section 4.3 describes how semantics enabled relevancy ranking is implemented in *C-Search*. In Section 4.4, we show how *C-Search* can be efficiently implemented using the inverted index technology.

4.1 From Words to Complex Concepts

Searching documents, in *C-Search*, is implemented using complex concepts expressed in a propositional Description Logic (DL) [5] language L^C (i.e., a DL language without roles). Complex concepts are computed by analyzing meaning of words and phrases in queries and document bodies.

Single words are converted into atomic concepts uniquely identified as the *lemma-sn*, where *lemma* is the lemma of the word, and *sn* is the sense number in a lexical database such as WordNet [60]. For instance, the word *dog* used in the sense of a domestic dog, which is the first sense in the lexical database, is converted into the atomic concept *dog*-1. The conversion of words into concepts is performed as follows. First, we look up and enumerate all meanings of the word in the lexical database. Next, we perform word sense filtering, i.e., we discard word senses which are not relevant in the given context. In order to do this, we follow the approach presented in [98], which exploits part-of-speech (POS) tagging information and the lexical database for the disambiguation of words in short noun phrases. Differently from [98] we do not use the disambiguation technique which leaves only the most probable sense of the word, because of its low accuracy. If more than one sense is left after the word sense filtering step then we keep all the left senses. If no senses from the lexical database are found then *lemma* itself is used as the identifier for the atomic concept. In this case, *C-Search* is reduced to syntactic search.

Complex concepts are computed by extracting phrases and by analyzing their meaning. Noun phrases are translated into the logical conjunction of atomic concepts corresponding to the words in the phrase. For instance, the noun phrase *A little dog* is translated into the concept *little*-4 $\sqcap$ *dog*-1. Here, we adopt the approach described in [36] by defining the extension

of a concept as a set of noun phrases which *describe* this concept. For instance, the extension of the concept *little*-4 $\sqcap$ *dog*-1 is the set of all noun phrases about dogs of a small size.

Concepts in natural language can be described ambiguously. For instance, the phrase *A little dog or a huge cat* represents a concept which encodes the fact that it is unknown whether the *only* animal described in the document is *a little dog* or *a huge cat*. In order to support complex concepts which encode uncertainty (partial information) that comes from the coordination conjunction *OR* in natural language, we introduce the notion of *descriptive phrase*. We define a descriptive phrase as a set of noun phrases, representing alternative concepts, connected by *OR*:

$$descriptive\_phrase ::= noun\_phrase \ \{OR \ noun\_phrase\} \tag{4.1}$$

Descriptive phrases are translated into the logical disjunction of the formulas corresponding to the noun phrases. For instance, the phrase *A little dog or a huge cat* is translated into the concept (*little*-4 $\sqcap$ *dog*-1) $\sqcup$ (*huge*-1 $\sqcap$ *cat*-1). To locate descriptive phrases we, first, follow a standard NLP pipeline to locate noun phrases, i.e., we perform sentence detection, tokenization, POS tagging, and noun phrase chunking. Second, we locate descriptive phrases satisfying Formula 4.1.

In *C-Search*, every document d is represented as an enumerated sequence of conjunctive components $\sqcap A^d$ (where A^d is an atomic concept from d, e.g., *dog*-1) possibly connected by disjunction symbol "$\sqcup$". For example, in Figure 4.1 we show the sequences of $\sqcap A^d$ extracted from documents in Figure 1.1. Rectangles in Figure 4.1 represent either conjunctive components $\sqcap A^d$ or the disjunction symbol "$\sqcup$". A number in a square at the left side of a rectangle represents the *position* of the rectangle in the whole sequence. Note, that symbol "$\sqcup$" is used to specify that conjunctive components $\sqcap A^d$ connected by this symbol form a single disjunctive concept

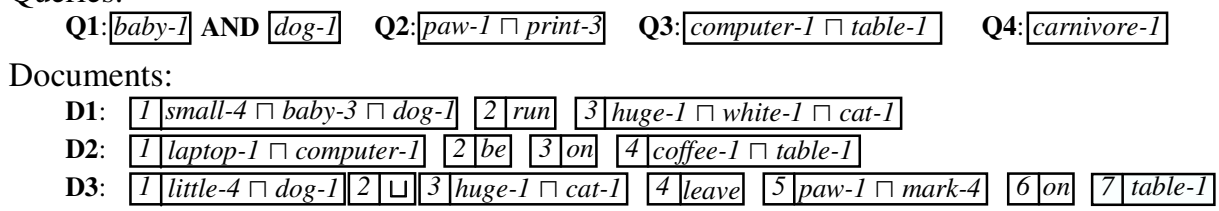


Figure 4.1: Document and Query Representations

$\sqcup \sqcap A^d$, namely:

$$\sqcup \sqcap A^d ::= (\sqcap A^d)\{(\text{“}\sqcup\text{”})(\sqcap A^d)\} \tag{4.2}$$

For example, the first three positions in the sequence for document $D3$ in Figure 4.1 represent the concept $(little\text{-}4 \sqcap dog\text{-}1) \sqcup (huge\text{-}1 \sqcap cat\text{-}1)$.

Queries usually are short phrases (i.e., 1-3 words) and, as shown in [98], standard NLP technology, primarily designed to be applied on full-fledged sentences, is not effective enough in this application scenario. For instance, an atomic concept in a query can be computed incorrectly, because of the selection of a wrong part-of-speech tag. In order to address this problem, for *short queries*, we use a POS-tagger which is specifically trained on short phrases [98]. On the other hand, for *long queries* (i.e., 4 words or more), we use the standard NLP technology.

Even if atomic concepts are computed correctly, complex concepts can be erroneously computed. One of the reasons is that a complex concept can be represented as a sequence of words without following the grammar for noun phrases. For instance, the query *cat huge* is converted into two atomic concepts *cat*-1 and *huge*-1, while the correct concept might be $cat\text{-}1 \sqcap huge\text{-}1$. Another reason is that a query describing more than one concept, without properly separating them, can be recognized as a single complex concept. For instance, the query *dog cat* is converted into the concept $dog\text{-}1 \sqcap cat\text{-}1$, while the user might be actually looking for a document describing both animals, i.e., *dog*-1 and *cat*-1. The examples described

above show that, in general, it is unknown how atomic concepts A_1^q, ..., A_n^q, extracted from short queries, should be combined in order to build complex query concepts. To represent this uncertainty we use the following query:

$$(A_1^q \text{ AND } \ldots \text{ AND } A_n^q) \text{ AND } (A_1^q \sqcup \cdots \sqcup A_n^q) \tag{4.3}$$

where the first part (A_1^q AND ... AND A_n^q), i.e., atomic concepts A_1^q, ..., A_n^q connected by using boolean operator AND, encodes the fact that it is *known* that the query answer should contain documents which are relevant to all the atomic concepts in the query. The second part, i.e., the complex concept $A_1^q \sqcup \cdots \sqcup A_n^q$, can be equivalently rewritten as $(A_1^q) \sqcup (A_2^q) \sqcup \cdots \sqcup (A_1^q \sqcap A_2^q) \sqcup \cdots \sqcup (A_1^q \sqcap \cdots \sqcap A_n^q)$ and, therefore, encodes the fact that it is *unknown* to which complex concept (e.g., $A_1^q \sqcap A_2^q$, or $A_1^q \sqcap \cdots \sqcap A_n^q$) the documents in the query answer should actually be relevant to. For instance, for queries *cat huge* and *dog cat* the following C-Search queries will be generated:

$$cat\ huge \Rightarrow cat\text{-}1 \text{ AND } huge\text{-}1 \text{ AND } cat\text{-}1 \sqcup huge\text{-}1 \sqcup (cat\text{-}1 \sqcap huge\text{-}1)$$
$$dog\ cat \Rightarrow dog\text{-}1 \text{ AND } cat\text{-}1 \text{ AND } dog\text{-}1 \sqcup cat\text{-}1 \sqcup (dog\text{-}1 \sqcap cat\text{-}1)$$

Note that in *C-Search* (as it will be discussed later in Section 4.3) we give a preference to documents which match complex concepts, therefore, in the first example, documents about $cat\text{-}1 \sqcap huge\text{-}1$ will be ranked higher. Let us assume that in the second example there are no documents about complex concept $dog\text{-}1 \sqcap cat\text{-}1$. In this case, it can be shown that the query results will be the same as the results of the query *dog*-1 AND *cat*-1.

4.2 From Word to Concept Matching

In *C-Search*, we allow the search of documents describing concepts which are semantically related to query concepts. We assume that, when a user is searching for a concept, she is also interested in more specific concepts[1]. For example, the extension of concept $(\mathit{little}\text{-}4 \sqcap \mathit{dog}\text{-}1) \sqcup (\mathit{huge}\text{-}1 \sqcap \mathit{cat}\text{-}1)$ is a subset of the extension of concept *carnivore*-1. Therefore, documents describing the former concept should be returned as answers to the query encoded by the latter concept. Formally a query answer $A(C^q, \mathcal{T})$ is defined as follows:

$$A(C^q, \mathcal{T}) = \{d \mid \exists C^d \in d, \text{ s.t. } \mathcal{T} \models C^d \sqsubseteq C^q\} \tag{4.4}$$

where C^q is a complex query concept extracted from the query q, C^d is a complex document concept extracted from the document d, and $\mathcal{T}$ is a terminological knowledge base (the background knowledge) which is used in order to check if C^d is more specific then C^q. Equation 4.4 states that the answer to a query concept C^q is the set of all documents d, such that, there exists concept C^d in d which is more specific than the query concept C^q.

During query processing we need to compute $A(C^q, \mathcal{T})$ for every query concept C^q in the query. One approach is to sequentially iterate through each concept C^d, compare it to the query concept C^q using semantic matching [38], and collect those C^d for which semantic matching returns *more specific* ($\sqsubseteq$). However, this approach may become prohibitory expensive as there may be thousands and millions of concepts described in documents. In order to allow for a more efficient computation of $A(C^q, \mathcal{T})$, we propose an approach described below.

[1]This could be easily generalized to any set of semantically related concepts. The impact of this choice onto the system performance is part of the future work.

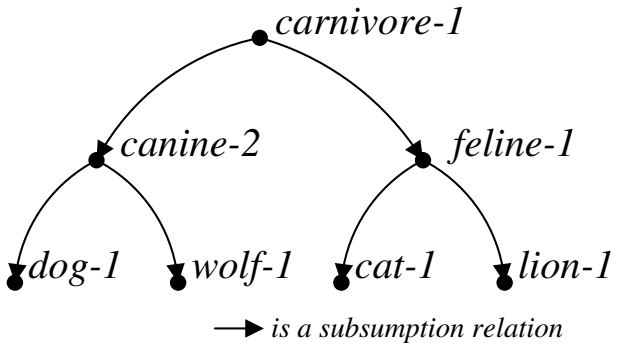


Figure 4.2: Example of terminological knowledge base $\mathcal{T}_{WN}$

Let us assume, as it is the case in the current implementation, that $\mathcal{T}$ consists of the terminological knowledge base $\mathcal{T}_{WN}$ generated from WordNet and extended by words (represented as concepts) for which no senses in WordNet are found. One small fragment of $\mathcal{T}_{WN}$ is represented in Figure 4.2. $\mathcal{T}_{WN}$ can be thought of as an acyclic graph, where links represent subsumption axioms in the form $A_i \sqsubseteq A_j$, with A_i and A_j atomic concepts.

Concepts C^d and C^q, are created by translating descriptive phrases into propositional DL formulas (see Section 4.1 for details). The resulting concepts are disjunctions ($\sqcup$) of conjunctions ($\sqcap$) of atomic concepts (A) without negation, i.e., $C^d \equiv \sqcup \sqcap A^d$ and $C^q \equiv \sqcup \sqcap A^q$. For example, possible document and query concepts are:

$$C^d \equiv (\mathit{little}\text{-}4 \sqcap \mathit{dog}\text{-}1) \sqcup (\mathit{huge}\text{-}1 \sqcap \mathit{cat}\text{-}1)$$
$$C^q \equiv (\mathit{small}\text{-}4 \sqcap \mathit{canine}\text{-}2) \sqcup (\mathit{large}\text{-}1 \sqcap \mathit{feline}\text{-}1)$$

By substituting C^d with $\sqcup \sqcap A^d$, C^q with $\sqcup \sqcap A^q$, and $\mathcal{T}$ with $\mathcal{T}_{WN}$ in Equation 4.4, we obtain:

$$A(\sqcup \sqcap A^q, \mathcal{T}_{WN}) = \{d \mid \exists(\sqcup \sqcap A^d) \in d, \text{ s.t. } \mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q\} \quad (4.5)$$

Let us denote by $\mathbf{C}_{\sqcup \sqcap A^q}$ the set of all the complex document concepts $\sqcup \sqcap A^d$, which are equivalent to or more specific than $\sqcup \sqcap A^q$, in formulas:

$$\mathbf{C}_{\sqcup \sqcap A^q} = \{\sqcup \sqcap A^d \mid \mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q\} \quad (4.6)$$

Then Equation 4.5 can be rewritten as follows:

$$A(\sqcup \sqcap A^q, \mathcal{T}_{WN}) = \{d \mid \exists(\sqcup \sqcap A^d) \in d, \text{ s.t. } (\sqcup \sqcap A^d) \in \mathbf{C}_{\sqcup \sqcap A^q}\} \quad (4.7)$$

In order to compute set $\mathbf{C}_{\sqcup \sqcap A^q}$, as defined in Equation 4.6, we need to solve the following subsumption problem:

$$\mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \quad (4.8)$$

Given that $\mathcal{T}_{WN}$ consists only of subsumption axioms between atomic concepts, and that concepts $\sqcup \sqcap A^d$ and $\sqcup \sqcap A^q$ do not contain negations, the problem in Equation 4.8 can be reduced to the set of subsumption problems

$$\mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq A^q \quad (4.9)$$

This problem reduction is obtained by applying the following three equations[2]:

$$\mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \text{ iff for all } \sqcap A^d \text{ in } \sqcup \sqcap A^d, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \quad (4.10)$$

$$\mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \text{ iff there exists } \sqcap A^q \text{ in } \sqcup \sqcap A^q, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcap A^q \quad (4.11)$$

$$\mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcap A^q \text{ iff for all } A^q \text{ in } \sqcap A^q, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq A^q \quad (4.12)$$

Notice that the second part of each equation is the same as the first part of the equation that follows, and that the first part of Equation 4.10 and the last part of Equation 4.12 are exactly Equations 4.8 and 4.9. This proves that the above problem reduction is correct.

If by $\mathbf{C}_C^{\sqcap}$ we denote a set of all the conjunctive components $\sqcap A^d$, which are equivalent to or more specific than concept C, i.e.,

$$\mathbf{C}_C^{\sqcap} = \{\sqcap A^d \mid \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq C\}, \text{ where } C \in \{A^q, \sqcap A^q, \sqcup \sqcap A^q\} \quad (4.13)$$

Given Equations 4.10, 4.11, and 4.12, the query answer $A(C^q, \mathcal{T}_{WN})$, as defined in Equation 4.7, can be computed by using Algorithm 1. The

[2]In Appendix A, we prove correctness and completeness of Equation 4.11 when the knowledge base and complex concepts are as described above. Note, that in general, Equation 4.11 cannot be applied. One such case is when negation is allowed, a counterexample is $\models A_i \sqsubseteq A_j \sqcup \neg A_j$. A second case is when $\mathcal{T}$ contains axioms of the form $A_i \sqsubseteq A_j \sqcup A_k$; consider, e.g., $\mathcal{T} = \{A_i \sqsubseteq A_j \sqcup A_k\} \models A_i \sqsubseteq A_j \sqcup A_k$.

Algorithm 1 Compute $A(\sqcup \sqcap A^q, \mathcal{T}_{WN})$

1: $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q} \leftarrow \emptyset$
2: **for** all $\sqcap A^q$ in $\sqcup \sqcap A^q$ **do**
3: $\quad i \leftarrow 0$
4: $\quad \mathbf{C}^{\sqcap}_{\sqcap A^q} \leftarrow \emptyset$
5: $\quad$ **for** all A^q in $\sqcap A^q$ **do**
6: $\quad\quad \mathbf{C}^{\sqcap}_{A^q} \leftarrow \{\sqcap A^d \,|\, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq A^q\}$
7: $\quad\quad$ **if** i = 0 **then**
8: $\quad\quad\quad \mathbf{C}^{\sqcap}_{\sqcap A^q} \leftarrow \mathbf{C}^{\sqcap}_{A^q}$ — Phase 1
9: $\quad\quad$ **else**
10: $\quad\quad\quad \mathbf{C}^{\sqcap}_{\sqcap A^q} \leftarrow \mathbf{C}^{\sqcap}_{\sqcap A^q} \cap \mathbf{C}^{\sqcap}_{A^q}$ — Phase 2
11: $\quad\quad$ **end if**
12: $\quad\quad i \leftarrow i + 1$
13: $\quad$ **end for**
14: $\quad \mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q} \leftarrow \mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q} \cup \mathbf{C}^{\sqcap}_{\sqcap A^q}$ — Phase 3
15: **end for**
16: $\mathbf{C}_{\sqcup \sqcap A^q} \leftarrow \{\sqcup \sqcap A^d \mid$ all $\sqcap A^d$ in $\sqcup \sqcap A^d$ belong to $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}\}$ — Phase 4
17: $A \leftarrow \{d \mid$ there exists $\sqcup \sqcap A^d$ in d, s.t., $\sqcup \sqcap A^d$ belongs to $\mathbf{C}_{\sqcup \sqcap A^q}\}$ — Phase 5

algorithm consists of the five principle phases which are described below:

Phase 1 (line 6) We compute $\mathbf{C}^{\sqcap}_{A^q}$, i.e., the set of all $\sqcap A^d$, such that, $\sqcap A^d \sqsubseteq A^q$.

Phase 2 (lines 5-13) We compute $\mathbf{C}^{\sqcap}_{\sqcap A^q}$, i.e., a set of all $\sqcap A^d$, such that, $\sqcap A^d \sqsubseteq \sqcap A^q$. As it follows from Equation 4.12, $\sqcap A^d \in \mathbf{C}^{\sqcap}_{\sqcap A^q}$ only if for every A^q in $\sqcap A^q$, $\sqcap A^d \in \mathbf{C}^{\sqcap}_{A^q}$. To compute $\mathbf{C}^{\sqcap}_{\sqcap A^q}$, we intersect sets $\mathbf{C}^{\sqcap}_{A^q}$ for all A^q in $\sqcap A^q$.

Phase 3 (lines 2-15) We compute the set $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$, i.e., the set of all $\sqcap A^d$, such that, $\sqcap A^d \sqsubseteq \sqcup \sqcap A^q$. As it follows from Equation 4.11, $\sqcap A^d \in \mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$ if $\sqcap A^d \in \mathbf{C}^{\sqcap}_{\sqcap A^q}$ at least for one $\sqcap A^q$ in $\sqcup \sqcap A^q$. To compute the set $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$, we take the union of all the sets $\mathbf{C}^{\sqcap}_{\sqcap A^q}$ for all $\sqcap A^q$ in $\sqcup \sqcap A^q$.

Phase 4 (line 16) We compute the set $\mathbf{C}_{\sqcup \sqcap A^q}$, i.e., the set of all complex

document concepts $\sqcup \sqcap A^d$, such that, $\sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q$. As it follows from Equation 4.10, $\sqcup \sqcap A^d \in \mathbf{C}_{\sqcup \sqcap A^q}$ only if all the conjunctive components $\sqcap A^d$ in $\sqcup \sqcap A^d$ belong to $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$. To compute the set $\mathbf{C}_{\sqcup \sqcap A^q}$, we collect all $\sqcup \sqcap A^d$ which consist only from conjunctive components $\sqcap A^d$ in $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$.

Phase 5 (line 17) We compute $A(\sqcup \sqcap A^q, \mathcal{T}_{WN})$ as defined in Equation 4.7, i.e., by collecting all the documents which contain concepts from $\mathbf{C}_{\sqcup \sqcap A^q}$.

The next section reports different approaches to how all the phases of the above algorithm can actually be implemented and their output on a running example.

In *C-Search*, query concepts C^q can be combined into more complex queries q by using the boolean operators AND, OR, and NOT. Query answer $A(q, \mathcal{T}_{WN})$ in this case is computed by recursively applying the following rules:

$$\begin{aligned} A(q_i \text{ AND } q_j, \mathcal{T}_{WN}) &= A(q_i, \mathcal{T}_{WN}) \cap A(q_j, \mathcal{T}_{WN}) \\ A(q_i \text{ OR } q_j, \mathcal{T}_{WN}) &= A(q_i, \mathcal{T}_{WN}) \cup A(q_j, \mathcal{T}_{WN}) \\ A(q_i \text{ NOT } q_j, \mathcal{T}_{WN}) &= A(q_i, \mathcal{T}_{WN}) \ / \ A(q_j, \mathcal{T}_{WN}) \end{aligned} \tag{4.14}$$

For instance, the query answer for query *baby*-1 AND *dog*-1 (in Figure 4.1) is computed as follows: $A(\textit{baby}\text{-1 AND } \textit{dog}\text{-1}, \mathcal{T}_{WN}) = A(\textit{baby}\text{-1}, \mathcal{T}_{WN}) \cap A(\textit{dog}\text{-1}, \mathcal{T}_{WN}) = \emptyset \cap \{D1, D3\} = \emptyset$

4.3 Relevance Ranking

In order to compute the relevance of documents, in *C-Search*, standard IR ranking techniques are adapted. In syntactic search, ranking of documents is usually performed by calculating frequencies $f(w^q, d)$ of appearance of

query words w^q in a document d and then by applying different scoring functions $score(w^q, d) = score(f(w^q, d), d)$, which depend on $f(w^q, d)$ and, in general, can also depend on many other parameters, e.g., length of d (see Section 1.1). In order to adapt such techniques for ranking query results in *C-Search*, $f(w^q, d)$ is replaced by the following function:

$$f'(A^q, w^q, d) = P(A^q, w^q) \cdot \sum_{A^d \sqsubseteq A^q} SS(A^q, A^d) \cdot P(A^d, w^d) \cdot f(A^d, w^d, d) \quad (4.15)$$

which takes into account frequencies $f(A^d, w^d, d)$ of all the atomic document concepts A^d which are related to A^q and described in d. Moreover, $f'(A^q, w^q, d)$ takes into account the fact that not all the atomic concepts A^q are equally important to query concept A^q. To measure this importance the following three parameters are used: $SS(A^q, A^d)$ - a measure of semantic similarity between concepts A^d and A^q; $P(A^q, w^q)$ - a coefficient which is proportional to the probability of that atomic concept A^q is correctly assigned to a word w^q in the query; and $P(A^d, w^d)$ - a coefficient which is proportional to the probability that an atomic concept A^d is correctly assigned to a word w^d in the document. Informally, $f'(A^q, w^q, d)$ is higher for: (i) concepts A^d which are closer in the meaning to the concept A^q, (ii) a concept A^q which is more likely to be a correct sense of a word w^q in a query q, and (iii) concepts A^d which are more likely to be correct senses for words w^d in a document d.

As a measure of semantic similarity $SS(A^q, A^d)$ the following formula is used[3]:

$$SS(A^q, A^d) = \frac{1}{10^{dist(A^q, A^d)}} \quad (4.16)$$

where $dist(A^q, A^d)$ is a distance between concepts A^q and A^d in the concept hierarchy from $\mathcal{T}_{WN}$. To estimate the coefficients $P(A, w)$, we use the

[3]Note that other measures of semantic similarity (e.g., see [16]) can also be used. It is a part of our future work to analyze the performance of different semantic similarity measures.

following formula:

$$P(A, w) = \frac{freq(A, w)}{maxFreq(w)} \tag{4.17}$$

where $freq(A, w)$ is a number provided by WordNet which shows how frequently the specified word w is used to represent the meaning A (incremented by one in order to avoid zero values), $maxFreq(w)$ is a maximum $freq(A, w)$ for w.

As an example, let us compute value $f'(A^q, w^q, d)$ for a concept *canine*-2 in a document $D1$ from Figure 4.1. The only more specific concept for a concept *canine*-2 in $D1$ is a concept *dog*-1. Given that the distance $dist(canine\text{-}2, dog\text{-}1)$ is equal to one (see Figure 4.2), $SS(canine\text{-}2, dog\text{-}1)$ will be equal to 10^{-1}. Assume that concepts *canine*-2 and *dog*-1 are the only concepts assigned to words *canine* and dog respectively. In this case, $P(canine\text{-}2, canine) = P(dog\text{-}1, dog) = 1$. A word *dog* appears in document $D1$ only once, therefore, $f(dog\text{-}1, dog, D1) = 1$. Finally, $f'(canine\text{-}2, canine, D1) = 1 * 10^{-1} * 1 * 1 = 10^{-1}$.

If by $score'(q, d)$ we denote a relevance score (for the document d with respect to the query q) computed by a syntactic relevancy ranking technique with $f(w^q, d)$ replaced by $f'(A^q, w^q, d)$, then the final document score with respect to the complex query concept $\sqcup \sqcap A^q$ is computed by using the following equation.

$$score(\sqcup \sqcap A^q, d) = score'(q, d) \cdot \sum_{\sqcap A^q \in \sqcup \sqcap A^q} isAns(\sqcap A^q, d) \cdot size^2(\sqcap A^q) \tag{4.18}$$

where $isAns(\sqcap A^q, d)$ is a function which has value 1 when document d describes at least one conjunctive component $\sqcap A^d$ which is more specific than the given conjunctive component $\sqcap A^q$ from the query concept $\sqcup \sqcap A^q$; otherwise, it has value 0. Function $size(\sqcap A^q)$ returns the number of atomic concepts in conjunctive component $\sqcap A^q$. Informally, $score(\sqcup \sqcap A^q, d)$ is higher for those documents d which are answers to more complex concepts $\sqcap A^q$ in $\sqcup \sqcap A^q$.

4.4 Concept Search via Inverted Indexes

In the section, we will show how document representations (e.g., see Figure 4.1) can be indexed and retrieved by using (different modifications of) inverted indexes.

4.4.1 Approach 1: C-Search via a Record Level Inverted Index

In this section, we describe how the document representations (see Figure 4.1) can be indexed and retrieved by using a record level inverted index (as it was proposed in [34]). In the inverted index, as used in syntactic search (see Section 1.2), there are two parts: the *dictionary*, i.e., a set of terms (t) used for indexing; and a set of posting lists P(t). A posting list P(t) is a list of all the postings for term t:

$$P(t) = [\langle d, freq \rangle]$$

where $\langle\, d,\, freq \rangle$ is a posting consisting of a document d associated with term t and the frequency $freq$ of t in d.

In *Approach 1*, inverted indexes are used to:

1. index conjunctive components $\sqcap A^d$ by their atomic concepts A^d. We call the resulting index the *concept* $\sqcap$*-index*. Concept $\sqcap$-index stores a mapping from each atomic concept to a set of all the conjunctive components which contain this concept. In Figure 4.3, we show a fragment of a concept $\sqcap$-index.

2. index complex concepts $\sqcup \sqcap A^d$ by their conjunctive components $\sqcap A^d$. We call the resulting index the *concept* $\sqcup$*-index*. Concept $\sqcup$-index stores a mapping from each conjunctive component $\sqcap A^d$ to a set of complex concepts $\sqcup \sqcap A^d$ which contain this component. In Figure 4.4, we show a fragment of a concept $\sqcup$-index.

Dictionary (t)	Posting lists (P(t))
little-4	$[\langle \textit{little-4} \sqcap \textit{dog-1}, 1\rangle; \langle \textit{small-4} \sqcap \textit{baby-1} \sqcap \textit{dog-1}, 1\rangle]$
dog-1	$[\langle \textit{little-4} \sqcap \textit{dog-1}, 1\rangle; \langle \textit{small-4} \sqcap \textit{baby-1} \sqcap \textit{dog-1}, 1\rangle]$
huge-1	$[\langle \textit{huge-1} \sqcap \textit{cat-1}, 1\rangle; \langle \textit{huge-1} \sqcap \textit{white-1} \sqcap \textit{cat-1}, 1\rangle]$
cat-1	$[\langle \textit{huge-1} \sqcap \textit{cat-1}, 1\rangle; \langle \textit{huge-1} \sqcap \textit{white-1} \sqcap \textit{cat-1}, 1\rangle]$

Figure 4.3: Concept $\sqcap$-index

Dictionary (t)	Posting lists (P(t))
little-4 $\sqcap$ *dog-1*	$[\langle (\textit{little-4} \sqcap \textit{dog-1}) \sqcup (\textit{huge-1} \sqcap \textit{cat-1}), 1\rangle]$
huge-1 $\sqcap$ *cat-1*	$[\langle (\textit{little-4} \sqcap \textit{dog-1}) \sqcup (\textit{huge-1} \sqcap \textit{cat-1}), 1\rangle]$

Figure 4.4: Concept $\sqcup$-index

3. index documents by (complex) concepts described in the documents. We call the resulting index the *document index*. The document index stores a mapping from each (complex) concept to a set of all the documents which describe this concept. In Figure 4.5, we show a fragment of a document index.

Now we will see how Algorithm 1 in Section 4.2 can be implemented in *Approach 1* given that concept $\sqcap$- and $\sqcup$- indexes as well as document index were constructed.

Phase 1 To compute the set $\mathbf{C}^{\sqcap}_{A^q}$ (line 6 in Algorithm 1), first, we search the knowledge base $\mathcal{T}_{WN}$ for a set $\mathbf{C}^{A}_{A^q}$ of atomic concepts A which are equivalent to or more specific than A^q. For example (see Figure 4.2),

$$\mathbf{C}^{A}_{\textit{canine-2}} = \{\textit{canine-2}, \textit{dog-1}, \textit{wolf-1}, \dots\}$$
$$\mathbf{C}^{A}_{\textit{feline-1}} = \{\textit{feline-1}, \textit{cat-1}, \textit{lion-1}, \dots\}$$
$$\mathbf{C}^{A}_{\textit{little-4}} = \{\textit{little-4}, \dots\}$$

Second, we collect all the conjunctive components $\sqcap A^d$ in $\mathbf{C}^{\sqcap}_{A^q}$ by

Dictionary (t)	Posting lists (P(t))
small-4⊓*baby-1*⊓*dog-1*	$[\langle D1, 1\rangle]$
huge-1⊓*white-1*⊓*cat-1*	$[\langle D1, 1\rangle]$
(*little-4*⊓*dog-1*) ⊔ (*huge-1*⊓*cat-1*)	$[\langle D3, 1\rangle]$

Figure 4.5: Document index

searching in the concept ⊓-index with atomic concepts from $\mathbf{C}^{A}_{A^q}$. For instance (see Figure 4.3),

$$\mathbf{C}^{\sqcap}_{canine\text{-}4} = \{little\text{-}4 \sqcap dog\text{-}1, small\text{-}4 \sqcap baby\text{-}1 \sqcap dog\text{-}1\}$$
$$\mathbf{C}^{\sqcap}_{feline\text{-}1} = \{huge\text{-}1 \sqcap cat\text{-}1, huge\text{-}1 \sqcap white\text{-}1 \sqcap cat\text{-}1\}$$
$$\mathbf{C}^{\sqcap}_{little\text{-}4} = \{little\text{-}4 \sqcap dog\text{-}1, small\text{-}4 \sqcap baby\text{-}1 \sqcap dog\text{-}1\}$$

Phase 2 Compute the set $\mathbf{C}^{\sqcap}_{\sqcap A^q}$ (lines 5-13 in Algorithm 1). For instance,

$$\mathbf{C}^{\sqcap}_{little\text{-}4 \sqcap canine\text{-}1} = \{little\text{-}4 \sqcap dog\text{-}1, small\text{-}4 \sqcap baby\text{-}1 \sqcap dog\text{-}1\}$$

Phase 3 Compute the set $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$ (lines 2-15 in Algorithm 1). For instance,

$$\begin{aligned}\mathbf{C}^{\sqcap}_{canine\text{-}2 \sqcup feline\text{-}1} =& \{little\text{-}4 \sqcap dog\text{-}1, small\text{-}4 \sqcap baby\text{-}1 \sqcap dog\text{-}1,\\ & huge\text{-}1 \sqcap cat\text{-}1, huge\text{-}1 \sqcap white\text{-}1 \sqcap cat\text{-}1\}\end{aligned}$$

Phase 4 We compute the set $\mathbf{C}_{\sqcup \sqcap A^q}$ (line 16 in Algorithm 1) by searching in the concept ⊔-index with conjunctive components from $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$. Note that we search only for those concepts $\sqcup \sqcap A^d$ which have all their conjunctive components $\sqcap A^d$ in $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$ and discard other concepts. For instance (see Figure 4.4),

$$\begin{aligned}\mathbf{C}_{canine\text{-}2 \sqcup feline\text{-}1} =& \{small\text{-}4 \sqcap baby\text{-}3 \sqcap dog\text{-}1, huge\text{-}1 \sqcap white\text{-}1 \sqcap cat\text{-}1,\\ & (little\text{-}4 \sqcap dog\text{-}1) \sqcup (huge\text{-}1 \sqcap cat\text{-}1)\}\end{aligned}$$

Phase 5 The query answer (line 17 in Algorithm 1) is computed by searching in the document index with complex concepts from $\mathbf{C}_{\sqcup\sqcap A^q}$. For instance (see Figure 4.5),

$$A(canine\text{-}2 \sqcup feline\text{-}1, \mathcal{T}_{WN}) = \{D1, D3\}$$

The described above approach has several potential problems. First, the size of an inverted index dictionary in concept $\sqcup$-index and document index, in the worst case, is exponential with respect to the size of terminology $\mathcal{T}_{WN}$. Second, the search time can be lengthy. If the query concepts A^q are very general, than, in phases 1,4, and 5, the sets $\mathbf{C}^A_{A^q}$, $\mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q}$, and $\mathbf{C}_{\sqcup\sqcap A^q}$ can contain many (in principle, all) related concepts. Consequently, the search time, which is growing linearly with the number of related (complex) concepts, can exceed an acceptable limit.

4.4.2 Approach 2: C-Search via a Word Level Inverted Index

In this section, we describe how the document representations (see Figure 4.1) can be indexed and retrieved by using a positional (word level) inverted index (as it was proposed in [35]). In a positional inverted index, differently from a record level inverted index, a posting list P(t) additionally contains all the positions of term t within a document.

$$P(t) = [\langle d, freq, [position]\rangle]$$

where $\langle\, d,\ freq,\ [position]\rangle$ is a posting consisting of a document d associated with term t, the frequency $freq$ of t in d, and a list $[position]$ of positions of t in d.

In *Approach 2*, we adopt a positional inverted index to index conjunctive components $\sqcap A^d$ by all more general or equivalent atomic concepts from $\mathcal{T}_{WN}$. For example, in Figure 4.6 we show a fragment of the positional inverted index created by using the document representations in Figure 4.1.

Dictionary (t)	Posting lists (P(t))
$\sqcup$	$[\langle D3, 1, [2]\rangle]$
baby-3	$[\langle D1, 1, [1]\rangle]$
canine-2	$[\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$
carnivore-1	$[\langle D1, 2, [1, 3]\rangle; \langle D3, 2, [1, 3]\rangle]$
computer-1	$[\langle D2, 1, [1]\rangle]$
feline-1	$[\langle D1, 1, [3]\rangle; \langle D3, 1, [3]\rangle]$
leave	$[\langle D3, 1, [4]\rangle]$
little-4	$[\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$

Figure 4.6: Positional Inverted Index

The inverted index dictionary, in *Approach 2*, consists of atomic concepts from $\mathcal{T}_{WN}$ (e.g., concepts *baby*-3 and *canine*-2 in Figure 4.6), and symbol "$\sqcup$" (e.g., the first term in Figure 4.6). Note that differently from *Approach 1*, the size of the dictionary in this case is the same as the size of $\mathcal{T}_{WN}$. The posting list $P(A)$ for an atomic concept A stores the positions of conjunctive components $\sqcap A^d$, such that, $\sqcap A^d \sqsubseteq A$. For instance, P(*canine-2*) $= [\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$, which means that at first position in documents $D1$ and $D3$ there are conjunctive components (i.e., *small*-4 $\sqcap$ *baby*-3 $\sqcap$ *dog*-1 and *little*-4 $\sqcap$ *dog*-1) which are more specific than *canine-2*. The posting list $P(\sqcup)$ stores the positions of the symbol "$\sqcup$".

Now, let us see how Algorithm 1 in Section 4.2 can be implemented by using the positional information of conjunctive components $\sqcap A^d$ stored in the inverted index. Notice that below instead of conjunctive components themselves we work only with their positions in documents.

Phase 1 Positions of conjunctive components $\sqcap A^d$ in the set $\mathbf{C}^{\sqcap}_{A^q}$ (line 6 in Algorithm 1) are computed by fetching the posting list $P(A^q)$ for an atomic concept A^q. For instance (see Figure 4.6),

$$\mathbf{C}^{\sqcap}_{\textit{little-4}} = [\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$$
$$\mathbf{C}^{\sqcap}_{\textit{carnivore-1}} = [\langle D1, 2, [1, 3]\rangle; \langle D3, 2, [1, 3]\rangle]$$

Phase 2 The intersection of the sets of conjunctive components (line 10 in Algorithm 1) is implemented by the intersection of corresponding posting lists. For instance,

$$\mathbf{C}^{\sqcap}_{little\text{-}4 \sqcap carnivore\text{-}1} = [\langle D1, 1, [1] \rangle; \langle D3, 1, [1] \rangle]$$

Phase 3 The union of the sets of conjunctive components (line 14 in Algorithm 1) is implemented by uniting corresponding posting lists. For instance,

$$\mathbf{C}^{\sqcap}_{canine\text{-}2 \sqcup feline\text{-}1} = [\langle D1, 2, [1, 3] \rangle; \langle D3, 2, [1, 3] \rangle]$$

Phase 4 Every concept in set $\mathbf{C}_{\sqcup \sqcap A^q}$ (line 16 in Algorithm 1) should consists only from the conjunctive components in $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$. In order to find the positions of such concepts, we take the union of the posting lists for $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$ with the posting list for the symbol "$\sqcup$". Then we filter out all the positions which does not comply with the pattern defined in Equation 4.2. For instance, for complex query concept $canine\text{-}2 \sqcup feline\text{-}1$, we will find the following complex document concepts:

$$\langle D1, 1, [1] \rangle \Rightarrow \boxed{1 \mid small\text{-}4 \sqcap baby\text{-}3 \sqcap dog\text{-}1}$$
$$\langle D1, 1, [3] \rangle \Rightarrow \boxed{3 \mid huge\text{-}1 \sqcap white\text{-}1 \sqcap cat\text{-}1}$$
$$\langle D3, 1, [1, 2, 3] \rangle \Rightarrow \boxed{1 \mid little\text{-}4 \sqcap dog\text{-}1} \boxed{2 \mid \sqcup} \boxed{3 \mid huge\text{-}1 \sqcap cat\text{-}1}$$

Phase 5 The query answer (line 17 in Algorithm 1) is computed by collecting the documents from all the postings. For instance,

$$A(canine\text{-}2 \sqcup feline\text{-}1, \mathcal{T}_{WN}) = \{D1, D3\}$$

If n is a number of atomic concepts A^q in the query concept $\sqcup \sqcap A^q$, then to compute $A(C^q, \mathcal{T}_{WN})$ it takes n posting list merges (i.e., intersections and unions). Note that, in a positional inverted index, the same number

of posting list merges is required to process a phrase query consisting of $n + 1$ words [59].

The main problem with *Approach 2*, is that the size of the index can be relatively big. If s is the maximum number of concepts which are assigned to each word after the word sense filtering step and *depth* is the depth of the knowledge base $\mathcal{T}_{WN}$ (i.e., the number of edges on the longest path from the concept to a root concept in $\mathcal{T}_{WN}$), then, in the worst case, a semantic inverted index in *Approach 2* will contain $s * depth$ times more postings than an inverted index in syntactic search for the same document collection. For example, if $depth = 16$ (as it is the case in WordNet) and $s = 3$, then, in the worst case, semantic inverted index will contain 48 times more postings than the syntactic inverted index.

4.4.3 Approach 3: C-Search with a Minimum Index Size

In this section, we propose *Approach 3* which is a modification of *Approach 2*, such that, the size of an inverted index is minimized.

First, in *Approach 3*, positions of conjunctive components are indexed only by atomic concepts which are contained in the conjunctive components $\sqcap A^d$ (and not by all the more general atomic concepts as it was done in *Approach 2*). Algorithm 1, in this case, is implemented in the same way as in *Approach 2* apart from *phase 1*. Now, in *phase 1*, we first search the knowledge base $\mathcal{T}_{WN}$ for a set $\mathbf{C}^A_{A^q}$ of atomic concepts A which are equivalent to or more specific than A^q. Second, the positions of conjunctive components $\sqcap A^d$ in the set $\mathbf{C}^{\sqcap}_{A^q}$ (line 6 in Algorithm 1) are computed by fetching the posting lists $P(A^q)$ for atomic concepts A^q in $\mathbf{C}^A_{A^q}$ and merging them.

Second, all the atomic concepts which have been assigned for a word, in *Approach 3*, are stored in a single posting (and not in separate postings as it was done in *Approach 2*). An inverted index which supports payloads

is used. A payload is metadata that can be stored together with each occurrence of a term[4]. In a positional inverted index with payloads, a posting list P(t) is represented as follows:

$$P(t) = [\langle d, freq, [position, payload]\rangle]$$

where *payload* is a sequence of bytes of an arbitrary length which is associated with the term t and which can be used to codify additional information about t at the position *position* in the document d.

The inverted index dictionary, in *Approach 3*, consists only of word lemmas (as in syntactic search) and the symbol "␣". Payloads are used to store sense numbers *sn* in WordNet. Each payload is seen as a bit array, where the size of the array is equal to the number of possible senses for a given lemma and the positions of bits which are set to one are used to represent active sense numbers *sn* (note that in general all the bits can be set to one if no senses were filtered out). For instance, if we take the word *dog* which has 7 senses in WordNet, then the posting list $P(dog)$ created by using the document representations in Figure 4.1 will be as follows:

$$P(dog) = [\langle D1, 1, [1, 1000000]\rangle; \langle D3, 1, [1, 1000000]\rangle]$$

During the retrieval, the posting list P(A^q =*lemma-sn*) can be computed, first, by fetching posting list P(*lemma*) and then by filtering out all the positions where *sn* bit is not set to one.

In this approach, the size of the inverted index dictionary as well as the number of postings are almost the same as in the inverted index in syntactic search (note that the symbol "␣" is the only additional term). Payloads take some additional space, but it can also be minimized. For instance, we can store a payload in a posting only when its value is different from the one in the preceding posting and use the payload value from the

[4] `http://lucene.apache.org/java/2_4_0/api/org/apache/lucene/index/Payload.html`

Table 4.1: Statistics for the number of more specific concepts

Number **N** of more specific concepts	Number of concepts with **N** more specific concepts	Number of concepts (%)
N<=10	95264	94.98
10<N<=100	4130	4.12
100<N<=1000	745	0.74
1000<N<=10000	136	0.13
10000<N<=100000	28	0.03

preceding posting otherwise.

$$P(dog) = [\langle D1, 1, [1, 1000000]\rangle; \langle D3, 1, [1]\rangle]$$

Here the assumption is that the same word tends to have the same meaning within the same document [28]. If it is the case, then only a few additional bytes will be stored for each document.

Similarly to *Approach 1*, the potential problem of *Approach 3* is that the search time can increase if we search for a very general atomic concept A^q. Note, however, that differently from *Approach 1*, we need to consider only related atomic concepts, and not all the complex concepts, which, in the worst case, are exponentially many.

4.4.4 Approach 4: C-Search with a Hybrid Index

Approach 3 can perform well if atomic concepts A^q in a query have only a few more specific concepts in $\mathcal{T}_{WN}$, but it can become inefficient otherwise. In Table 4.1, we show a statistics for a number of more specific concepts in WordNet. As we can observe from Table 4.1, only 909 out of 100303 concepts (i.e., less than 1%) are very general, i.e., have more than 100 more specific atomic concepts. For instance, concepts *mammal-1*, *animal-1*, and *entity-1* have 9472, 12953, and 76439 more specific atomic concepts respectively. These 909 concepts form a tree (where $\top$ is a root) which we call a *'cap'* of the knowledge base.

In this section, we propose *Approach 4* which combines Approaches 2 and 3, where *Approach 2* is used only for concepts in the *cap* and *Approaches 3* is used for the rest of the concepts. Let us consider how it is done in detail. First, all the concepts are indexed by using *Approach 2*. Second, each concept which is not in the cap is additionally indexed by the most specific concept(s) from the cap, which is more general than the given concept.

During the retrieval, in order to compute the posting list $P(A^q)$ for a concept A^q which is in the cap, first, we follow *Approach 3*, where the set $\mathbf{C}^A_{A^q}$ consists only of atomic concepts A which are equivalent to or more specific than A^q and are in the cap. Second, we follow *Approach 2* by using the most specific atomic concepts from $\mathbf{C}^A_{A^q}$ (i.e., we use only those concepts which don't have more general concepts in $\mathbf{C}^A_{A^q}$). The results of both approaches are then merged. For concepts which are not in the cap we just follow *Approach 3*. Note that for all the concepts inside/outside the cap *Approach 3* is used for a relatively small number of atomic concepts.

If s is the number of senses, then in the worst case the index will contain $2 * s$ times more postings (and not $s * depth$ as in *Approach 2*) than in the syntactic search approach. Moreover, the search time in *Approach 4* can be always kept relatively small for both very specific and very general concepts (which is not the case in *Approach 3*).

4.4.5 Approach 5: Approximated C-Search

As it was discussed in Section 4.3, only those atomic document concepts A^d are scored high which are not very distant from the query concepts A^q (see Formula 4.15) in the concept hierarchy of $\mathcal{T}_{WN}$. Therefore, if we use only the closest concepts, the quality of results returned by *C-Search* should not be affected much. Moreover, as it was discussed in Sections 4.4.1-4.4.4, by using fewer related concepts, we can decrease the search time.

Approach 5 is a modified version of *Approach 3* which approximates the results of *C-Search* by using not all but only more specific concepts within distance *dist* from the atomic concept A^q. Also, in *Approach 5*, we limit the number of atomic concepts which can be assigned for each word in a query, by selecting only the s most probable senses. The influence of parameters *dist* and s on a quality and a performance of *C-Search* is discussed in Section 9.1.5.

4.5 Summary

In this chapter we presented a novel approach to document retrieval in which syntactic search is extended with a layer of semantics which enables semantic searching still fully reusing the proven IR technologies such as the inverted index. The *C-Search* approach can be positioned anywhere in the semantic continuum (see Figure 2.1) with the purely syntactic search being its base case, and the full semantic search being the optimal solution, at the moment beyond the available technology.

To the best of our knowledge, there are few approaches that are based on similar ideas to those of *C-Search*. For example, Hybrid Search [10] is similar to *C-Search* in that it combines syntactic search with semantic search. Differently from us, in [10], semantic search is implemented on metadata and is totally separated from syntactic search, implemented on keywords. Another approach, reported in [18], uses classes and instances of an RDF ontology to annotate documents in order to combine ontology-based search with the classical IR. Our approach is different from both [10] and [18] in that it is based on a seamless *integration* of syntactic and semantic kinds of search within a single solution enabled by the proven IR technology based on inverted indexes. In a sense, instead of using a reasoning engine to enable semantics (as it is done e.g. in [18]), we integrated

semantic reasoning within an inverted index, by taking advantage of the simplifying assumptions that we made about the ontologies used to enable the semantic search (see Section 4.2).

Chapter 5

Document Classification: Get-Specific Algorithm

Classification hierarchies have always been a natural and effective way for humans to organize their knowledge about the world. These hierarchies are rooted trees where each node defines a topic category. Child nodes' categories define aspects or facets of the parent node's category, thus creating a multifaceted description of the objects which can be classified in these categories. Classification hierarchies are used pervasively: in conventional libraries (e.g., the Dewey Decimal Classification system (DDC) [19]), in web directories (e.g., DMoz[1]), in e-commerce standardized catalogues (e.g., UNSPSC[2]), and so on.

Standard classification methodologies amount to manually organizing objects into classification categories following a predefined system of rules. The rules may differ widely in different approaches, but there is one classification pattern which is commonly followed. The pattern is called the *get-specific principle*, and it requires that an object is classified in a category (or in a set of categories), which most specifically describes the object. Following this principle is not easy and is constrained by a number of lim-

[1] `http://dmoz.org/`

[2] `http://www.unspsc.org/`

itations, discussed below:

- the meaning of a given category is implicitly codified in a natural language label, which may be ambiguous and may therefore be interpreted differently by different classifiers;
- a link, connecting two nodes, may also be ambiguous in the sense that it may specify the meaning of the child node, of the parent node, or of both;
- as a consequence of the previous two items, the classification task also becomes ambiguous in the sense that different classifiers may classify the same objects differently, based on their subjective opinion.

In this chapter we describe the approach which address the three problems discussed above by implementing the *get-specific principle* through propositional reasoning on complex concepts extracted from classification labels. Material presented in this chapter has been developed in collaboration with Fausto Giunchiglia and Ilya Zaihrayeu and published in [39].

The remainder of the chapter is organized as follows. In Section 5.1 we introduce the classification model, we show how the get-specific algorithm can be described in this model, and we identify the main problems peculiar to the algorithm. In Section 5.2 we show how classifications can be translated into formal classifications, how the get-specific algorithm can be encoded in the concept language, and how its peculiar problems can be dealt with in the concept language.

5.1 The Get-Specific Algorithm

In this section, we describe the *get-specific* document classification algorithm.

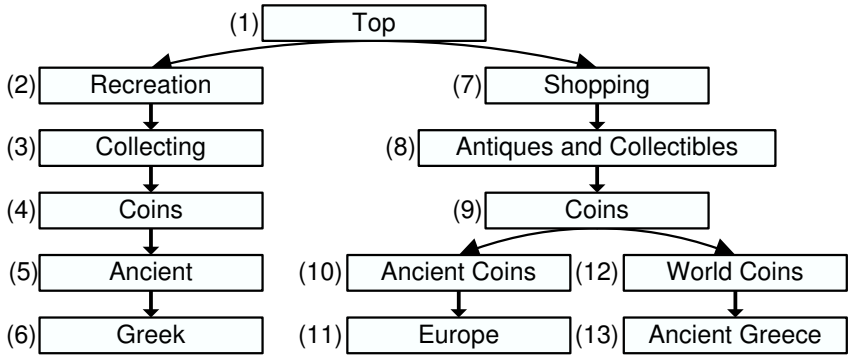


Figure 5.1: A part of the DMoz web directory

5.1.1 Classifications and a Classification Model

To avoid ambiguity of interpretation, in Definition 5.1.1 we formally define the notion of classification; and in Figure 5.1 we give an example of a classification, extracted from the DMoz web directory and adjusted for sake of presentation.

Definition 5.1.1. *A classification is a rooted tree* $C = \langle N, E, L \rangle$ *where* N *is a set of nodes,* E *is a set of edges on* N*, and* L *is a set of labels expressed in a natural language, such that for any node* $n_i \in N$*, there is one and only one label* $l_i \in L$*.*

We see the process of classification as a decision making procedure in which the classification tree is divided into a set of minimal decision making blocks. Each block consists of a node (called the root node of the block) and its child nodes (see Figure 5.2). While classifying an object, the classifier considers these blocks in a top-down fashion, starting from the block at the classification root node and then continuing to blocks rooted at those child nodes, which were selected for *further consideration.* These nodes are selected following decisions which are made at each block along two dimensions: *vertical* and *horizontal.* In the vertical dimension, the classifier decides which of the child nodes are selected as candidates for further consideration. In the horizontal dimension, the classifier decides which of

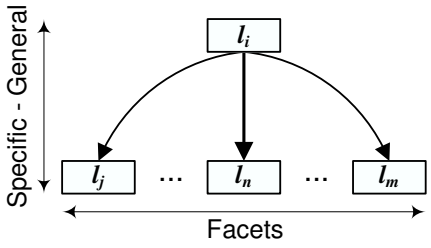


Figure 5.2: The decision making block

the candidates are *actually* selected for further consideration. If none of the child nodes are appropriate or if there are no child nodes, then the root node of the block becomes a *classification alternative* for the given object. The process reiterates and continues recursively until no more nodes are left for further consideration. At this point, all the classification alternatives are computed. The classifier then decides which of them are most appropriate for the classification of the object and makes *the final classification choice*.

5.1.2 Modelling the Get-Specific Classification Algorithm

In this subsection we discuss the general principles lying behind the get-specific algorithm and we show how these principles can be implemented within the model introduced in the previous subsection. Particularly, we discuss how vertical and horizontal choices, as well as the final classification choice are made.

- **Vertical choice.** Classification hierarchies are organized such that upper level categories represent more general concepts, whereas lower level categories represent more specific concepts. When the classifier searches for an appropriate category for the classification of an object, she looks for the ones which most specifically describe the object and, therefore, when making a vertical choice, she selects a child node as a candidate if it describes the object more specifically than the par-

ent does. For example, if a document about ancient Greek coins is classified in the classification from Figure 5.1, then node n_6 is more appropriate for the classification than node n_5. When this principle is applied recursively, it leads to the selection of the category which lies as deep in the classification hierarchy as possible. The principle described above is commonly called the *get-specific principle*. Let us consider, for instance, how Yahoo! describes it:

> *"When you suggest your site, get as specific as possible. Dig deep into the directory, looking for the appropriate subcategory."*[3]

- **Horizontal choice.** Child nodes may describe different aspects or *facets* of the parent node and, therefore, more than one child node may be selected in the vertical choice if a multifaceted document is being classified. As a consequence of this, the classifier needs to decide which of the several sibling nodes are appropriate for further consideration. When one sibling node represents a more specific concept than another, then the former is usually preferred over the latter. For example, node n_{10} is more appropriate for the classification of ancient Greek coins than node n_{12}. As a rule of thumb, the horizontal choice is made in favor of as few nodes as possible and, preferably, in favor of one node only. We call the principle described above, the *get-minimal principle*. Consider, for instance, how DMoz describes it.

> *"Most sites will fit perfectly into one category. ODP categories are specialized enough so that in most cases you should not list a site more than once."*[4]

[3]Yahoo! guidelines: See `http://docs.yahoo.com/info/suggest/appropriate.html`
[4]DMoz guidelines: See `http://dmoz.org/guidelines/site-specific.html`

- **Tradeoff between vertical and horizontal choices.** The two principles described above cannot always be fulfilled at the same time. Namely, if the vertical choice results in too many candidates, then it becomes hard to fulfill the principle of minimality in the horizontal choice. In order to address this problem, a tradeoff needs to be introduced between the two requirements, which usually means trading specificity in favor of minimality. The following is an example of a tradeoff rule used in DMoz:

  > *"If a site offers many different things, it should be listed in a more general category as opposed to listing it in many specialized subcategories."*[4]

- **The final classification choice.** When all classification alternatives are determined, the classifier confronts all of them in order to make her final classification choice. Note that now the choice is made not at the level of a minimal decision making block, but at the level of the whole classification. However, the classifier uses the same selection criteria as those used in the horizontal choice. For example, nodes n_6 and n_{13} are more appropriate for the classification of documents about ancient Greek coins than node n_{11}.

5.1.3 Problems of the Get-Specific Classification Algorithm

As discussed in [36], there are several problems which are common to document classification algorithms. The problems are caused by the potentially large size of classifications, by ambiguity in natural language labels and in document descriptions, by different interpretations of the meaning of parent-child links, and so on. All these problems lead to nonuniform, duplicate, and error-prone classification. In addition to the problems discussed in [36], the get-specific algorithm has two peculiar problems, related to the

two decision dimensions. We discuss these problems below on the example of a document titled *"Gold Staters in the Numismatic Marketplace"*, being classified in the classification from Figure 5.1.

- **Vertical choice: the "I don't know" problem.** The classifier may make a mistake because she does not (fully) understand the meaning of a child node or the relation of the document to that node, whereas the node is a valid candidate. For example, the classifier may not know that "Gold Stater" is a coin of ancient Greece and, therefore, will erroneously classify the document into node n_5, whereas a more appropriate node is n_6.

- **Horizontal choice: the "Polarity change" problem.** The classifier may make a mistake when one of the sibling candidate nodes is more appropriate for further consideration than another, but a descendent of the latter is more appropriate for the classification than a descendant of the former node. For instance, the label of node n_{10} more specifically describes the document than the label of node n_{12}. Therefore, the classifier will choose node n_{10} only as a candidate and will finally classify the document in node n_{11}, whereas a more appropriate node for the classification is node n_{13}, a descendent of n_{12}.

5.2 Formalizing the Get-Specific Algorithm

In this section we formalize the get-specific classification algorithm by encoding it as a problem expressed in L^C. First, we discuss how natural language node labels and document descriptions are converted into formulas in L^C. Second, we discuss how we reduce the problems of vertical, horizontal, and final classification choices to fully automated propositional

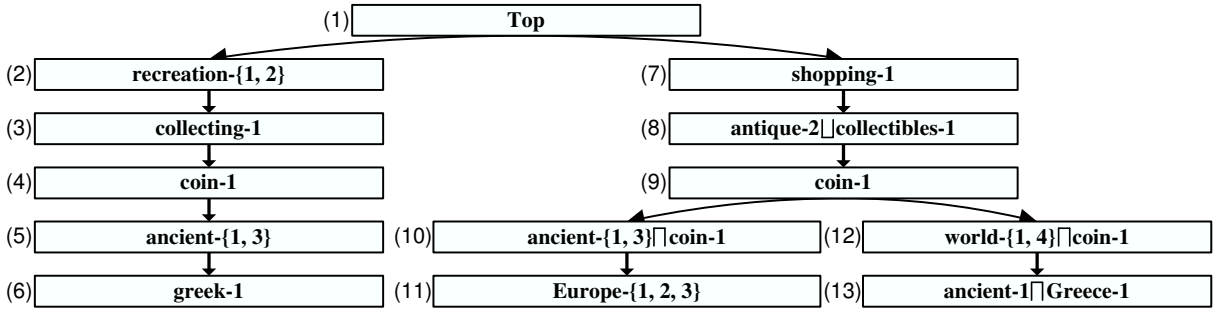


Figure 5.3: Formal Classification

reasoning. Finally, we show how the problems discussed in Section 5.1.3 can be dealt with in a formal way.

5.2.1 From Natural Language to Formal Language

Classification labels are expressed in a natural language, which is ambiguous and, therefore, is very hard to reason about. In order to address this problem, we encode classification labels into formulas in L^C following the approach proposed in [36]. This allows us to convert the classification into a new structure, which we call *Formal Classification* (FC):

Definition 5.2.1. *A Formal Classification is a rooted tree* $FC = \langle N, E, L^F \rangle$ *where* N *is a set of nodes,* E *is a set of edges on* N*, and* L^F *is a set of labels expressed in* L^C*, such that for any node* $n_i \in N$*, there is one and only one label* $l_i^F \in L^F$*.*

Note that natural language labels l_i (see Definition 5.1.1) are converted into formulas l_i^F by following the approach described in Section 4.1. For instance, in Figure 5.3 we show the result of conversion of the classification from Figure 5.1 into a FC.

Before a document can be automatically classified, it also has to be assigned an expression in L^C, which we call the document concept, written C^d. The assignment of a concept to a document is done in two steps:

first, a set of n keyphrases is retrieved from the document using text mining techniques (see, for example, [93, 49]); the keyphrases are converted to formulas in L^C, and the document concept is then computed as the conjunction of the formulas.

5.2.2 The Algorithm

In the following we describe how we make vertical and horizontal choices, compute the tradeoff, and make the final classification choice in FCs.

- **Vertical choice.** A child node n_i is a candidate, given that a document with concept C^d is being classified, if the label of the node, l_i^F, subsumes C^d, i.e., if the following holds: $C^d \sqsubseteq l_i^F$. In formulas, if N_c is the set of child nodes in the block, then we compute the vertical choice $V(C^d)$ as:

$$V(C^d) = \{n_i \in N_c | C^d \sqsubseteq l_i^F\} \tag{5.1}$$

  If the vertical choice results in no candidates, then root node n_r of the current block is added to the set of classification alternatives $A(C^d)$:

$$\textbf{if } |V(C^d)| = 0 \textbf{ then } A(C^d) \leftarrow A(C^d) \cup \{n_r\} \tag{5.2}$$

  In Figure 5.4a we show an example of a situation when two child nodes n_2 and n_4 are selected for further consideration, and in Figure 5.4b we show an example of a situation when no child node can be selected.

- **Horizontal choice.** Given the set of candidates $V(C^d)$, we exclude those nodes from the set, whose label is more general than the label of another node in the set. In formulas, we compute the horizontal choice $H(C^d)$ as:

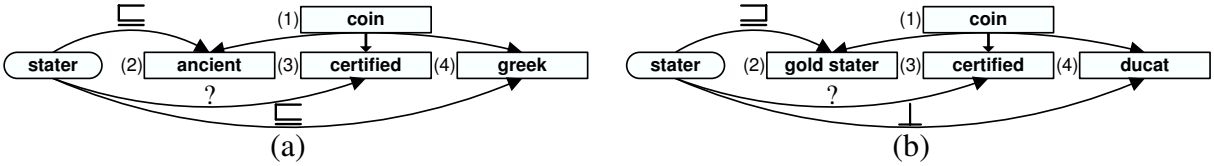


Figure 5.4: Vertical choice ("?" means no relation is found)

$$H(C^d) = \{n_i \in V(C^d) | \nexists n_j \in V(C^d),\ s.t.\ j \neq i, l_j^F \sqsubseteq l_i^F, and\ l_j^F \not\sqsupseteq l_i^F\} \tag{5.3}$$

We introduce the last condition (i.e., $l_j^F \not\sqsupseteq l_i^F$) to avoid mutual exclusion of nodes, whose labels in the FC are equivalent concepts. For instance, two syntactically different labels "seacoast" and "seashore" are translated into two equivalent concepts. When such situation arises, all the nodes, whose labels are equivalent, are retained in $H(C^d)$.

- **The tradeoff.** Whenever the size of $H(C^d)$ exceeds some threshold k, the nodes of $H(C^d)$ are discarded as candidates and root node n_r of the block is added to the set of classification alternatives $A(C^d)$. In formulas:

$$\textbf{if} |H(C^d)| > k \textbf{ then } H(C^d) \leftarrow \emptyset \textbf{ and } A(C^d) \leftarrow A(C^d) \cup \{n_r\} \tag{5.4}$$

- **The final classification choice.** When no more nodes are left for further consideration, set $A(C^d)$ includes all the classification alternatives. We compare them to make the final classification choice, but, differently from vertical and horizontal choices, we compare the *meanings* of nodes given their path to the root, and not their labels. We encode the meaning of node n_i into a concept in L^C, called *concept of*

node [38], written C_i^n, and computed as:

$$C_i^n = \begin{cases} l_i^F & \text{if } n_i \text{ is the root of the } FC \\ l_i^F \sqcap C_j^n & \text{if } n_i \text{ is not the root, where } n_j \text{ is the parent of } n_i \end{cases} \tag{5.5}$$

Similar to how the horizontal choice is made, we exclude those nodes from $A(C^d)$, whose concept is more general than the concept of another node in the set. In formulas, we compute the final classification choice $C(A)$ as:

$$C(A) = \{n_i \in A(C^d) | \nexists n_j \in A(C^d),\ s.t.\ j \neq i,\ C_j^n \sqsubseteq C_i^n,\ and\ C_j^n \not\sqsupseteq C_i^n\} \tag{5.6}$$

The last condition (i.e., $C_j^n \not\sqsupseteq C_i^n$) is introduced to avoid mutual exclusion of nodes with the same meaning in the classification hierarchy. For instance, two paths *top/computers/games/soccer* and *top/sport/soccer/computer_games* lead to two semantically equivalent concepts. When such situation arises, all the nodes with the same meaning are retained in $C(A)$.

Computing Equations 5.1, 5.3, and 5.6 requires verifying whether the subsumption relation holds between two formulas in L^C. As shown in [36], a problem expressed in L^C can be rewritten as an equivalent problem expressed in propositional logic. Namely, if we need to check whether a certain relation rel (which can be $\sqsubseteq$, $\sqsupseteq$, $\equiv$, or $\perp$) holds between two concepts A and B, given some knowledge base $\mathcal{KB}$ (which represents our a priori knowledge), we construct a propositional formula according to the pattern shown in Equation 5.7 and check it for validity:

$$\mathcal{KB} \rightarrow rel(A, B) \tag{5.7}$$

5.2.3 Dealing with Problems

Encoding a classification algorithm into a problem in L^C allows it to avoid many problems, which are common to classification algorithms [36]. Particularly, since the problem is encoded in a formal language, there is no ambiguity in interpretation of classification labels, of edges, and document contents. Apart from this, since computation is performed by a machine, the problem of classification size becomes largely irrelevant. Finally, since the formal algorithm is deterministic, the classification is always performed in a uniform way.

In Section 5.1.3 we discussed two problems, peculiar to the get-specific algorithm. Below we discuss what they mean in L^C and how they can be dealt with.

- **Vertical choice: the "I don't know" problem.** This problem arises when the specificity relation in Equation 5.1 cannot be computed while a human observes that it exists. The problem is caused by lack of background knowledge and it can be dealt with by adding missing axioms to the underlying knowledge base [37]. For instance, if we add a missing axiom which states that concept *Stater* (defined as "any of the various silver or gold coins of ancient Greece") is more specific than concept *Greek* (defined as "of or relating to or characteristic of Greece ..."), then the algorithm will correctly classify document *"Gold Staters in the Numismatic Marketplace"* into node n_6 in the classification shown in Figure 5.1.

- **Horizontal choice: the "Polarity change" problem.** The problem arises when the label of node n_i is more specific than the label of its sibling node n_j (i.e., $l_i^F \sqsubseteq l_j^F$), but the concept of a n_i's descendant node n_k is more general than the concept of a n_j's descendant node n_m (i.e., $C_k^n \sqsupseteq C_m^n$). In the simplest case, this problem can be dealt

with by not performing the horizontal choice. In this case, both n_k and n_m will be in the classification alternative set for some document, and n_k will then be discarded when the final classification choice is made.

5.3 Summary

In this chapter, first, we provided a classification model and showed how the get-specific document classification algorithm, commonly used in hierarchical document classification systems, can be described in this model. Second, we showed how the get-specific algorithm can be fully automated using a knowledge-centric approach, an approach which is conceptually different from the one used in Information Science.

Part III

Semantics Enabled P2P Search

Chapter 6

P2P Concept Search

In order to provide semantic search in P2P networks, we propose to extend the centralized version of *C-Search* to *P2P C-Search*. First, we extend the reasoning with respect to a single background knowledge $\mathcal{T}$ to the reasoning with respect to the background knowledge $\mathcal{T}_{P2P}$ which is distributed among all the peers in the network. Second, we extend the centralized inverted index (II) to distributed inverted index build on top of DHT. The idea is schematically represented in the equation below.

$$\boxed{\textit{C-Search} \xrightarrow{Knowledge(\mathcal{T} \rightarrow \mathcal{T}_{P2P}), Index(II \rightarrow DHT)} \textit{P2P C-Search}}$$

Material presented in this chapter has been developed in collaboration with Fausto Giunchiglia, Sheak Rashed Haider Noori, Dharanipragada Janakiram, and Harisankar Haridas and published in [33, 45].

The remainder of this chapter is organized as follows. In Section 6.1, we show how the distributed background knowledge $\mathcal{T}_{P2P}$ can be implemented on top of DHT. In Section 6.2, we describe how DHT can be used, in *P2P C-Search*, to provide an efficient distributed semantic indexing and retrieval.

6.1 Distributed Knowledge

To access the background knowledge $\mathcal{T}$, stored on a single peer, *C-Search* needs at least the following three methods:

getConcepts(W) returns a set of all the possible meanings (atomic concepts A) for word W. For example, *getConcepts(canine)* $\rightarrow$ {*canine-1 ('conical tooth'), canine-2 ('mammal with long muzzles')*}.

getChildren(A) returns a set of all the more specific atomic concepts which are directly connected to the given atomic concept A in $\mathcal{T}$. For example, with respect to $\mathcal{T}$ in Figure 4.2, *getChildren (carnivore-1)* $\rightarrow$ {*canine-2, feline-1*}.

getParents(A) returns a set of all the more general atomic concepts which are directly connected to the given atomic concept A in $\mathcal{T}$. For example, with respect to $\mathcal{T}$ in Figure 4.2, *getParents (dog-1)* $\rightarrow$ {*canine-2*}.

In order to provide access to background knowledge $\mathcal{T}_{P2P}$ distributed over all the peers in the P2P network, we create distributed background knowledge *DBK*. In *DBK*, each atomic concept A is identified by a unique concept ID (A_{ID}) which is composed from peer ID (P_{ID}), where peer is a creator of the atomic concept, and local concept ID in the Knowledge Base of the peer. Every atomic concept A is represented as a 3-tuple: $A = \langle A_{ID}, POS, GLOSS \rangle$, where A_{ID} is a concept ID; POS is a part of speech; and $GLOSS$ is a natural language description of A. In the rest of the chapter, for the sake of presentation, instead of the complete representation $\langle A_{ID}, POS, GLOSS \rangle$ we use just *lemma-sn*.

DBK is created on top of a DHT. Atomic concepts are indexed by words using the DHT 'put' operation, e.g., *put(canine,* {*canine-1, canine-2*}*)*. Moreover, every atomic concept is also indexed by related atomic concepts together with the corresponding relations. We use a modification of the DHT 'put' operation *put(A, B, Rel)*, which stores atomic concept B with

relation *Rel* on the peer responsible for (a hash of) atomic concept A, e.g., *put(canine-2, dog-1, '$\sqsubseteq$')*, *put(canine-2, carnivore-1, '$\sqsupseteq$')*.

After *DBK* has been created, *getConcepts(W)* can be implemented by using the DHT 'get' operation, i.e., getConcepts(W) = get(W). Both methods *getChildren(A)* and *getParents(A)* are implemented by using a modified DHT 'get' operation *get(A, Rel)*, i.e., *getChildren(A) = get(W, '$\sqsubseteq$')* and *getParents(A) = get(W, '$\sqsupseteq$')*. The operation *get(A, Rel)* finds a peer responsible for atomic concept A and retrieve only those atomic concepts B which are in relation *Rel* with A.

Let us now see how *DBK* can be bootstrapped. At the beginning we have one single peer in the P2P network and *DBK* is equivalent to the background knowledge $\mathcal{T}$ of this peer. For example, $\mathcal{T}$ can be created from WordNet. A new peer joining the P2P network bootstraps its own background knowledge from *DBK* by doing the following three steps. First, the peer computes a set of words which are used in the local document collection. Second, the peer downloads from *DBK* a set of all the atomic concepts which are associated with these words by using 'getConcepts' method. Finally, the peer downloads all the more general atomic concepts by recursively calling 'getParents' method.

Notice, that by extending peer's background knowledge $\mathcal{T}$ to *DBK* which stores $\mathcal{T}_{P2P}$, we are likely to have a higher coverage on words, atomic concepts, and relations. Therefore, we can enable semantics to a higher extend in the semantic continuum, e.g., when user types a word which is not present in her $\mathcal{T}$, she can use atomic concepts from background knowledge of other peers stored in *DBK*.

The potential problem with the above approach is that it can require a lot of messages to collect and keep up to date all the more general concepts for every concept on each peer. In order to address this problem, we propose to cache the content of *DBK* on a special peer which we call the

caching peer. We fix an ID in the DHT, and the *caching peer* is dynamically selected based on the range of IDs the peer is responsible for (i.e., the ID should belong to the ID space of the peer). The request for a part of the DBK first goes to the *caching peer* (whose ip address can be cached) and then if the *caching peer* is busy or unavailable the request is processed by using the *DBK*.

6.2 Indexing and Retrieval

The query answer defined in Equation 4.4, can be extended to the case of distributed search by taking into account that the document collection D_{P2P} is equivalent to the union of all the documents from all the peers in the network (where each document d is uniquely identified by a document ID) and also that background knowledge $\mathcal{T}_{P2P}$ is distributed among all the peers.

$$QA(C^q,\mathcal{T}_{P2P}) = \{d \in D_{P2P} \mid \exists C^d \in d, \text{s.t. } \mathcal{T}_{P2P} \models C^d \sqsubseteq C^q\} \tag{6.1}$$

Let us consider a subset $QA(C^q,\mathcal{T}_{P2P}, A)$ of the query answer $QA(C^q,\mathcal{T}_{P2P})$. $QA(C^q,\mathcal{T}_{P2P}, A)$ consists of documents d which contain at least one complex concept C^d which is more specific than the complex query concept C^q and contains atomic concept A.

$$\begin{aligned} QA(C^q,\mathcal{T}_{P2P},A) = \{d \in D_{P2P} \mid & \exists C^d \in d, \text{s.t. } \mathcal{T}_{P2P} \models C^d \sqsubseteq C^q \\ & \text{and } \exists A^d \in C^d, \text{s.t. } A^d = A\} \end{aligned} \tag{6.2}$$

For the sake of presentation, in the rest of the chapter we assume $C^q = \sqcap A^q$. If we denote by $C(A)$ a set of all atomic concepts A^d, which are equivalent to or more specific than concept A, i.e.,

$$C(A) = \{A^d \mid \mathcal{T}_{P2P} \models A^d \sqsubseteq A\} \tag{6.3}$$

then, it can be shown that, given Equation 6.2, the query answer $QA(C^q,\mathcal{T}_{P2P})$ can be computed as follows

$$QA(C^q,\mathcal{T}_{P2P}) = \bigcup_{A \in C(A^*)} QA(C^q,\mathcal{T}_{P2P},A) \tag{6.4}$$

where A^* is an arbitrarily chosen atomic concept A^q in conjunctive component $C^q = \sqcap A^q$

Given Equation 6.4, the query answer can be computed by using the algorithm described below. The algorithm takes complex query concept C^q as input and computes a query answer QA in six macro steps:

Step 1 Initialize the query answer: $QA = \emptyset$. Initialize auxiliary sets $\mathbf{C}_{ms} = \emptyset$ and $\mathbf{C}'_{ms} = \emptyset$.

Step 2 Select one atomic concept A from complex query concept C^q and add it to $\mathbf{C}_{ms}$.

Step 3 For every $A \in \mathbf{C}_{ms}$, repeat steps 4 and 5.

Step 4 Compute $QA(C^q, \mathcal{T}_{P2P}, A)$ and add it to QA.

Step 5 Compute the set of all more specific atomic concepts B which are directly connected to the given atomic concept A in $\mathcal{T}_{P2P}$ and add them to $\mathbf{C}'_{ms}$.

Step 6 If $\mathbf{C}'_{ms} \neq \emptyset$, then assign $\mathbf{C}_{ms} = \mathbf{C}'_{ms}$, $\mathbf{C}'_{ms} = \emptyset$ and repeat step 3.

Note that on step 2 atomic concept A can be selected arbitrarily. In order to minimize the number of iterations, we choose A with the smallest number of more specific atomic concepts. The smaller the number, the fewer times we need to compute $QA(C^q, \mathcal{T}_{P2P}, A)$ on step 3.

In the following, we, first, show how documents are indexed in *P2P C-Search*, and then we show how the described above algorithm can be implemented.

In *P2P C-Search*, complex concepts are computed in the same way as in *C-Search* (see Section 4.1). The only difference is that now if an atomic concept is not found in the local background knowledge $\mathcal{T}$, then $\mathcal{T}_{P2P}$ is queried instead. *P2P C-Search* also uses the same document representation as *C-Search* (see Figure 4.1).

After document representations are computed, the indexing of documents is performed as follows. Every peer computes a set of atomic concepts A which appear in the representations of peer's documents. For every atomic concept A, the peer computes a set of documents d which contain A. For every pair $\langle A, d\rangle$, the peer computes a set $S(d, A)$ of all the complex document concepts C^d in d, which contain A.

$$S(d, A) = \{C^d \in d \mid A \in C^d\} \tag{6.5}$$

For example, if d is document $D1$ in Figure 4.1 and A is equivalent to *dog-1*, then $S(d, A) = \{\textit{small-4} \sqcap \textit{baby-3} \sqcap \textit{dog-1}\}$. For every A, the peer sends document summaries corresponding to A, i.e., pairs $\langle d, S(d, A)\rangle$, to a peer p_A responsible for A in *DBK*. The peer p_A indexes these summaries using the local *C-Search* (Approach 3 from Section 4.4.3 is used). Overall, every peer in the network is responsible for some words and for some atomic concepts. Peers maintain the following information for their words and concepts:

1. For every word, the peer stores a set of atomic concepts (word senses) for this word.

2. For every atomic concept, the peers stores a set of direct more specific and more general atomic concepts.

3. Document summaries $\langle d, S(d, A)\rangle$ for all the atomic concepts A (for which the peer is responsible) are stored on the peer and indexed in the local *C-Search*.

Word senses	
canine	*canine-1*, *canine-2*
More specific concepts	
canine-2	*dog-1*, *wolf-1*
More general concepts	
canine-2	*carnivore-1*

C-Search index	
canine-2	$\langle D4, 1, [1] \rangle$
population-4	$\langle D4, 1, [1] \rangle$

Figure 6.1: Peer's information

An example of the information, which can be stored on the peer responsible for a single word *canine* and for a single atomic concept *canine-2*, is shown in Figure 6.1.

Now, let us see how different steps of the algorithm for computing the query answer are implemented in *P2P C-Search*:

Step 1 Peer p_I initiates the query process for complex query concept C^q.

Step 2 p_I selects A in C^q with the smallest number of more specific atomic concepts. C^q is propagated to the peer p_A responsible for A. On peer p_A, QA is initialized to an empty set and A is added to $\mathbf{C}_{ms}$.

Step 3 Steps 4 and 5 are repeated for every atomic concept B in $\mathbf{C}_{ms}$.

Step 4 p_A submits C^q to the peer p_B responsible for B. p_B receives the query concept C^q and locally (by using *C-Search*) computes the set $QA(C^q, \mathcal{T}_{P2P}, B)$. The results are sent back to p_A. Note that if $B{=}A$, then the query propagation is not needed. On receiving new results $QA(C^q, \mathcal{T}_{P2P}, B)$, p_A merges them with QA.

Step 5 Moreover, p_B also computes the set of atomic concepts which are more specific than B by querying locally stored (direct) more specific concepts (e.g., see 'More specific concepts' in Figure 6.1). The results

are also propagated to peer p_A where they are added to set $\mathbf{C}'_{ms}$. If $B{=}A$, then the set of more specific concepts are computed on p_A itself.

Step 6 If $\mathbf{C}'_{ms} \neq \emptyset$, then p_A assigns $\mathbf{C}_{ms} = \mathbf{C}'_{ms}$, $\mathbf{C}'_{ms} = \emptyset$ and repeats step 3. Otherwise the results are sent to the initiator peer p_I.

Note, that, in order to optimize query propagation, peer p_A can pre-compute addresses of peers p_B which are responsible for more specific concepts, and use DHT to locate such peers only when pre-computed information is outdated.

An example showing how the query answer $QA(C^q, \mathcal{T}_{P2P}, A)$ is computed is given in Figure 6.2. Peers, represented as small circles, are organized in a DHT, represented as a circle. A query consisting of a single query concept $C^q = \mathit{little}\text{-}4 \sqcap \mathit{canine}\text{-}2$ is submitted to peer P_I. Let us assume that atomic concept, *canine*-2 has smaller number of more specific atomic concepts than concept *little*-4. In this case, C^q is propagated to a peer $P_{\mathit{canine}\text{-}2}$, i.e., the peer responsible for atomic concept *canine*-2. The query propagation is shown as a firm line in Figure 6.2. $P_{\mathit{canine}\text{-}2}$ searches in a local *C-Search* index with C^q. No results are found in this case. $P_{\mathit{canine}\text{-}2}$ collects all the atomic concepts which are more specific than *canine*-2, i.e., atomic concepts *dog*-1 and $wolf$-1. Query concept C^q is propagated to peers $P_{\mathit{dog}\text{-}1}$ and $P_{wolf\text{-}1}$. $P_{wolf\text{-}1}$ finds no results while $P_{\mathit{dog}\text{-}1}$ finds document D_1. D_1 is an answer because it contains concept $\mathit{small}\text{-}4 \sqcap \mathit{baby}\text{-}3 \sqcap \mathit{dog}\text{-}1$ which is more specific than $\mathit{little}\text{-}4 \sqcap \mathit{canine}\text{-}2$. D_1 is sent to P_A. The propagation of the results is shown as a dash line in Figure 6.2. Both peers $P_{\mathit{dog}\text{-}1}$ and $P_{wolf\text{-}1}$ have no more specific concepts than *dog*-1 and $wolf$-1, therefore C^q is not propagated to any other peers. P_A sends the final result, i.e. D_1, to peer P_I.

Note that the deeper we go in propagating the query, the less precise can be the answer. For instance, the user searching for *canine*-2 might be more

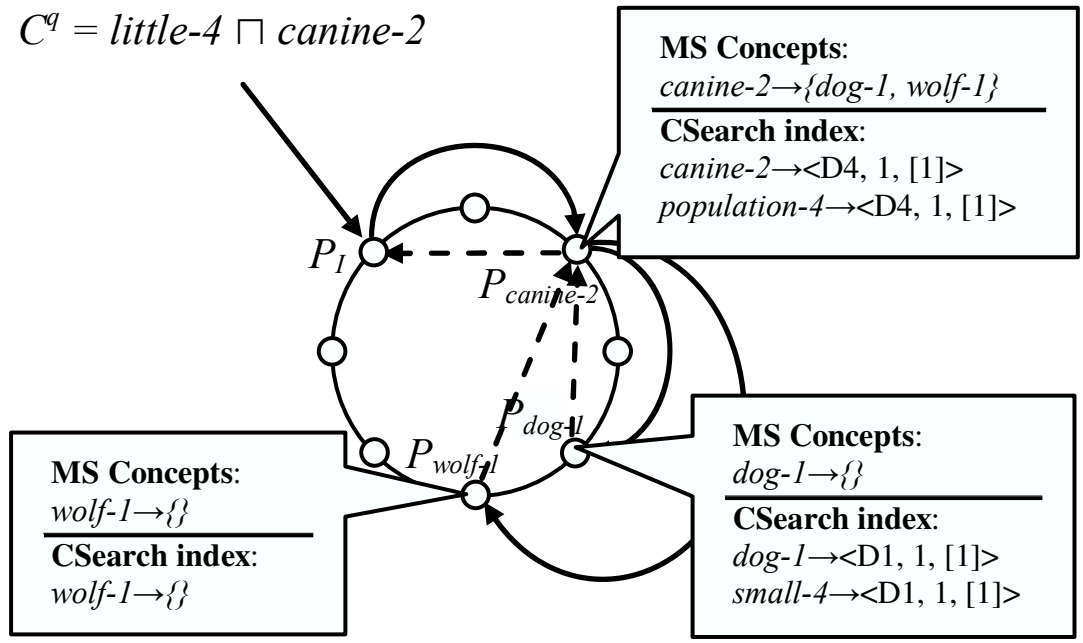


Figure 6.2: Query Answering

interested in documents about concept *canine*-2 than in documents about concept *dog*-1, and she may not be interested at all in documents about very specific types of dogs (e.g., $affenpinscher$-1). In *P2P C-Search*, we allow a user to specify the maximum allowed distance in terms of numbers of links between atomic concepts in $\mathcal{T}_{P2P}$. Notice that this distance is similar to a standard time-to-live (TTL) [71]. Moreover, we allow the user to specify the maximum number of more specific concepts which can be used per each atomic concept in C^q.

In order to compute the query answer for a more complex query, e.g., query *baby*-1 AND *dog*-1 (in Figure 4.1), the intersection of posting lists needs to be computed (see Equation 4.14). Since our approach is not replacing syntactic search but extending it with semantics, for an efficient implementation of the intersection, we can reuse the optimization techniques developed in P2P syntactic search (see e.g. Section 3.2).

6.3 Summary

In this chapter, we have presented an approach, called *P2P C-Search*, which allows for a semantic search on top of distributed hash table (DHT). There are two main aspects in which *P2P C-Search* extends *C-Search*: (i) centralized document index is replaced by distributed index build on top of DHT; (ii) reasoning with respect to a single background knowledge is extended to the reasoning with respect to the background knowledge distributed among all the peers in the network.

Chapter 7

Semantic Flooding

We can see the Web as a network of peers (a P2P network) where each peer stores various documents *about* a set of topics which are of interest to its users. Most often these documents are organized into classifications (see Section 5.1.1). An abstract example of user generated classifications of several peers can be seen in Figure 7.1. In this chapter, we show how multiple classifications can be exploited to help the user in finding documents about the topics which are the same or related to the topics (and/or queries) in the user's classification. For example, a user novice in some topic might benefit from finding a peer, the user of which is an expert in the given topic. Moreover, when searching for a particular document about the topic, even an expert might be interested in finding not only those documents which are stored in the local document collection, but also the documents stored on other peers.

In this chapter, we describe an approach, that we call *Semantic Flooding*, which is based on the following key ideas:

1. The first is that the links which connect nodes inside a classification together with the links which codify ontology mappings among multiple classifications form a *semantic overlay network* which can be exploited to perform a semantic search on nodes and later a semantic

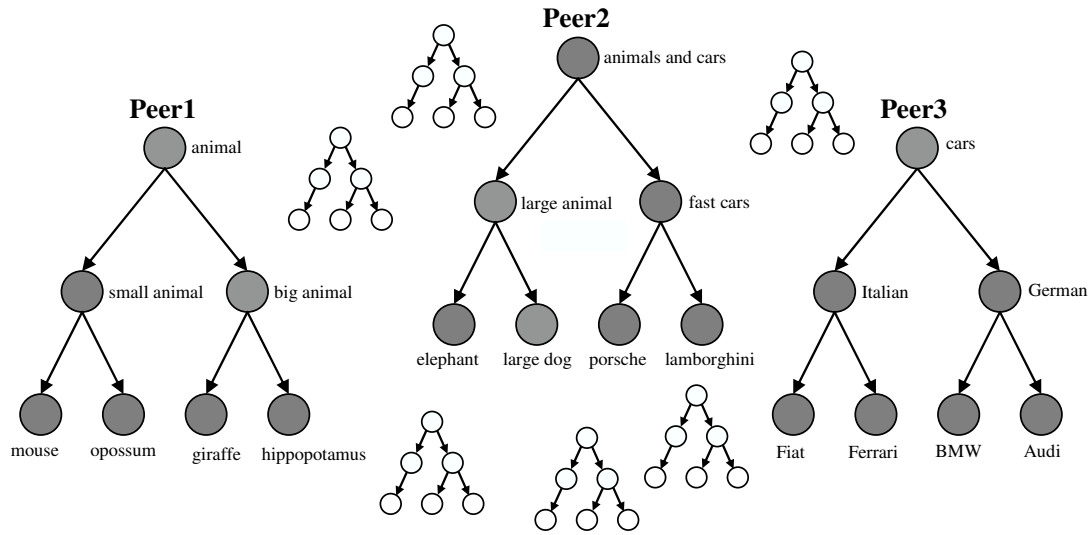


Figure 7.1: P2P Network of User-Generated Classifications

search on documents classified inside these nodes.

2. The second is that a semantic search is implemented by flooding the links of the semantic overlay network. Differently from "normal" flooding as it happens, for instance, in Gnutella[1], these links carry meaning and, more precisely, codify the semantic relation holding between two nodes (i.e., equivalence, more or less general) and allow, therefore, for "more informed" query propagation.

3. The third and last is that a semantic search inside a node is performed using *C-Search* (see Chapter 4), thus exploiting as much as possible the advantages of a syntactic search and also a semantic search, as a function of the available background knowledge [37].

Material presented in this chapter has been developed in collaboration with Fausto Giunchiglia, Alethia Hume, and Piyatat Chatvorawit and published in [32].

The chapter is structured as follows. In Section 7.1, we define the se-

[1] `http://en.wikipedia.org/wiki/Gnutella`

mantic overlay network built out of the classification links and mappings across classifications. In Section 7.2, we show how this network can be exploited to perform semantic flooding. In Section 7.3, we show how links across classifications can be computed via semantic matching (as described in [38]) or via *P2P Concept Search* (see Chapter 6).

7.1 A Semantic Overlay Network

In our approach, a user of each peer in the P2P network organizes her documents in a classification(s). Three examples of user-generated classifications are shown in Figure 7.1. Recall that nodes in the classification specify (complex) concepts which the user is interested in. For example, the user of *peer1* is interested in documents about *mice* and *hippopotamuses*. The whole classification specifies the user interest profile. For example, the user of *peer1* is interested in various kinds of animals, and the user of *peer3* is interested in cars. Notice, that a user can be interested in more than one topic. For instance, the user of *peer2* stores documents about both animals and cars.

In order to allow an automatic reasoning about classifications and their content, each classification is converted into Formal Classification (FC) and each document d is assigned a document concept C^d (see Section 5.2.1). To convert a classification into FC, the *background knowledge (BK)* [37] of the user is used. BK represents the knowledge of the user about concepts and their relationships over a specific domain or a limited set of domains. An example of a FC created from the classification in Figure 7.2a is shown in Figure 7.2b. After a FC has been created, documents can be automatically classified to nodes in the classification by using the get-specific principle (see Section 5.2). Formally, a set of documents $S(n)$ classified in a sub-tree

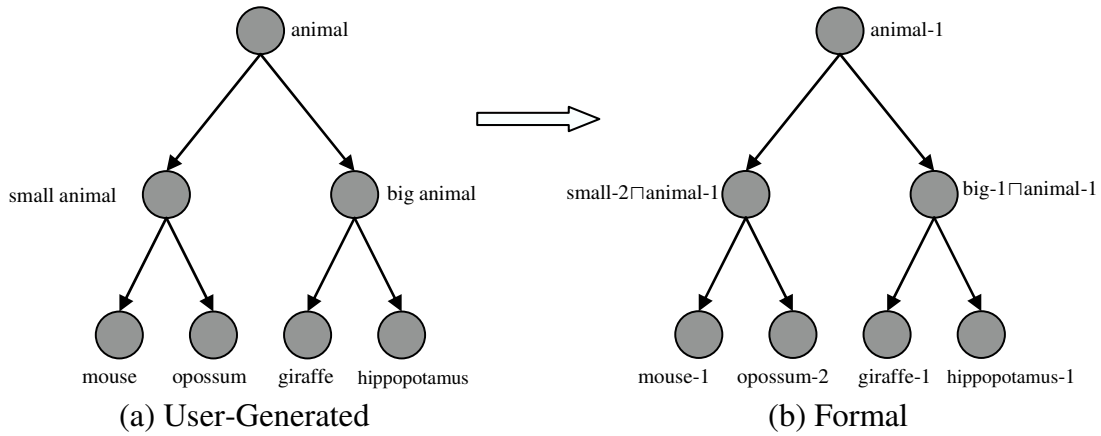


Figure 7.2: Classification

of node n is defined as follows:

$$S(n) = \{d \mid C^d \sqsubseteq C^n\} \tag{7.1}$$

where C^n is the concept of node (see Equation 5.5).

To make the peers in the P2P Network able to reason about the contents of each other, semantic links, expressed in the C-OWL language [15], can be created between related nodes in their classifications. We concentrate on the following links: (i) equivalence links $A \xrightarrow{\equiv} B$, (ii) more general links $A \xrightarrow{\sqsupseteq} B$, and (iii) more specific links $A \xrightarrow{\sqsubseteq} B$. For example, in Figure 7.3, the link between nodes with labels "large dog" and "huge dalmatian" is used to specify that the concept of the former node ($large$-1 $\sqcap$ dog-1) is more general than the concept of the latter node ($huge$-1 $\sqcap$ $dalmatian$-2). Note that according to the get-specific principle, all the documents which are classified in the subtree of the later node can be classified also in the subtree of the former node.

The set of the links which connect nodes inside a classification plus C-OWL links across classifications constitute a semantic overlay network which can be built on top of any underlying set of peers and their physical connections.

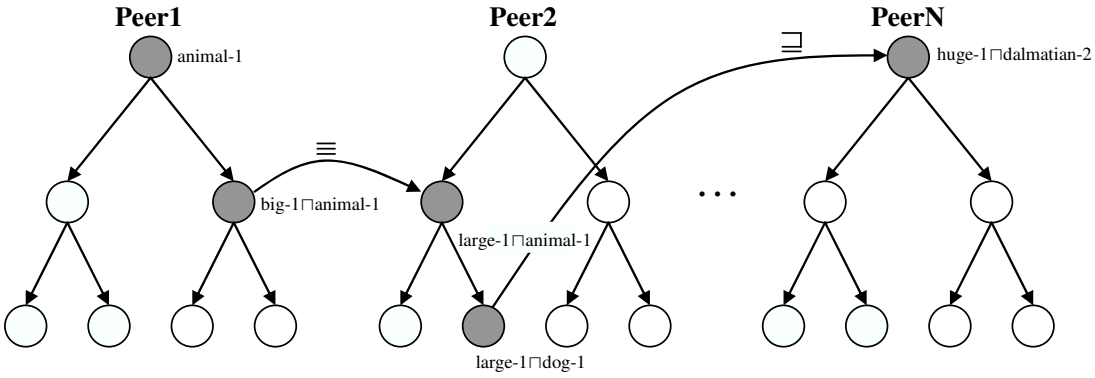


Figure 7.3: A Semantic Overlay Network

7.2 Semantic Flooding

When a user searches for documents, she, first, selects a node n in the classification. The root node of the classification serves as a default node for search if none of nodes are selected. Second, the user issues the query q. The query is converted into an expression in L^C as described in Section 4.1. Let C^n be a complex concept of node n and C^q be a complex concept extracted from query q. The goal of the *Semantic Flooding* algorithm is to find documents stored in the network, such that, concept of document C^d is more specific than the concept of node C^n and there exists concept C described in d which is more specific than the query concept C^q. Formally a query answer $A(C^n, C^q)$ is defined as follows:

$$A(C^n, C^q) = \{d \mid C^d \sqsubseteq C^n \text{ and } \exists C \in d, \text{s.t. } C \sqsubseteq C^q\} \tag{7.2}$$

The problem of a semantic search in the P2P network can be decomposed into three subproblems:

1. Identifying semantically relevant peers
2. Searching inside relevant peers
3. Aggregation of the search results

Let us consider these three subproblems in detail.

7.2.1 Identifying semantically relevant peers

We consider a peer to be semantically relevant to the query if there are nodes in peer's classification which are relevant to the node selected by the user. Moreover, some of the documents classified in these nodes should be relevant to the user query. In order to store the information about potentially relevant peers, the initiator peer p_I creates a peer information list, defined as follows:

$$peerinfos(n) = [\langle p, nodeinfos(p, n), stat\rangle],$$

where p is a relevant peer, $stat$ is a status of p: NQ - peer is not queried, QU - peer is already queried, or RE - response is returned, and $nodeinfos(p, n)$ is a list which stores information about nodes n' from peer p which are semantically related to node n plus a set $\{l\}$ of incoming links l for every node n':

$$nodeinfos(p, n) = [\langle n', \{l\}\rangle]$$

Initially, $peerinfos(n)$ contains information only about the local peer p_I: $peerinfos(n) = [\langle p_I, [\langle n, \emptyset\rangle], NQ\rangle]$. After $peerinfos(n)$ is initialized, p_I starts an infinite loop, where a single iteration is performed as follows:

- Select the first (if any) peer info $\langle p, nodeinfos(p, n), stat\rangle$ from $peerinfos(n)$, such that, $stat = NQ$.
- If there are no such peer infos, wait until the $peerinfos(n)$ list is modified and perform the previous step again.
- Form a query request $\langle C^n, C^q\rangle$ and submit it to peer p.
- Change the status of peer p to $stat = QU$.

When peer p receives the query request, it locally computes a set of links L, such that, a target node has a complex concept which is more specific than the complex concept C^n. It is important to note, that at the same time, the concept of the target node in a link can be equivalent, more specific, or more general than the concept of the source node. All the links in L are sent back to the initiator peer p_I. Peer p_I updates the $peerinfos(n)$ list by using information from links in L. $peerinfos(n)$ list is then sorted in a decreasing order of the number of incoming links. We assume that in this way, peers are queried in a decreasing order of their importance.

Every node n' in $nodeinfos(p, n)$ lists has only the documents with complex document concepts C^d which are more specific than the complex concept C^n. This is because, from $C^d \sqsubseteq C^{n'}$ and $C^{n'} \sqsubseteq C^n$, it follows that $C^d \sqsubseteq C^n$. In spite of this, links between nodes do not describe all complex concepts C, which can be found in the documents classified to these nodes. Therefore, it can be the case that node n' has no documents which are relevant to the query concept C^q. The portion of such nodes can increase when concept C^n becomes more and more general. In the worst case, i.e., when $C^n \equiv \top$, all the nodes which can be reached by all the links can be added to $nodeinfos(p, n)$ and all the corresponding peers p can be queried. Semantic flooding in this case is reduced to a normal flooding and, in general, can be very inefficient.

In order to implement a more efficient selection of semantically relevant peers, we propose to use a measure of semantic similarity $SS(C^{n'}, C^q)$ between complex concepts at node $C^{n'}$ and the complex query concept C^q (see, for example, [14]). As a simple example of a semantic similarity measure $SS(C^{n'}, C^q)$, let us consider the following measure:

$$SS(C^{n'}, C^q) = \begin{cases} 1 & \text{if } C^{n'} \sqsubseteq C^q \\ 0 & \text{otherwise} \end{cases}$$

Observe that for n' with $SS(C^{n'}, C^q) = 1$, concepts C^d, for all the documents classified to n', are more specific than query concept C^q. It is because, from $C^d \sqsubseteq C^{n'}$ and $C^{n'} \sqsubseteq C^q$, it follows that $C^d \sqsubseteq C^q$. Given that C^d is built from concepts C found in the document d, it is likely that d is relevant to query q. The following measure of semantic similarity is actually used[2]:

$$SS(C^{n'}, C^q) = \sum_{A^q \in C^q} \frac{1}{10^{\min\limits_{A^{n'} \in C^{n'}} (dist(A^{n'}, A^q))}} \tag{7.3}$$

Now, instead of just the number of incoming links, $peerinfos(n)$ list is sorted in a decreasing order of the peer scores computed as a sum of node scores $score(q, n')$. A node score $score(q, n')$ is computed as follows:

$$score(q, n') = (N_l + 1) * (SS(C^{n'}, C^n) + SS(C^{n'}, C^q)), \tag{7.4}$$

where, N_l is a number of incoming links for node n'. Note that only links for those nodes which are relevant for current search request are considered while sorting $peerinfos(n)$.

7.2.2 Searching inside a relevant peer

Each peer p, on receiving a search request $\langle C^n, C^q \rangle$, performs search for relevant documents in a local document collection by using the *C-Search* (see Chapter 4). The output of C-Search is a list of documents ordered by their relevance to the query. A list of top k ranked documents, nodes to which the documents are classified, and the information about frequencies of atomic concepts $A \in C^q$ in the retrieved documents and in the whole local document collection are sent back to the initiator peer p_I. Status of p is changed to $stat = RE$. In order to store the information about relevant documents, initiator peer p_I uses a document information list:

$$docinfos(q) = [\langle d, n', [\langle A, tf(A, d) \rangle]\rangle],$$

[2]Equation 7.3 is a generalized version of Equation 4.16

where d is a document which is classified to node n', and which is also relevant to query q, $tf(A, d)$ is a number which represents importance of document d to an atomic concept $A \in C^q$. Moreover, in order to store the global information about importance of atomic concepts $A \in C^q$, p_I uses term information lists for all A:

$$terminfos(A) = [\langle p, numDocs_p, docFreq_p(A)\rangle],$$

where $docFreq_p(A)$ is a number which represents the frequency of atomic concept A in the document collection of peer p which have $numDocs_p$ documents in total. On receiving new results, peer p_I updates the $docinfos(q)$ and $terminfos(A)$ tables.

The search process terminates when: (i) the required number (e.g., 100) of documents is retrieved; or (ii) all the relevant documents are retrieved; or (iii) the search time exceeds some predefined limits; or (iv) the user terminates the process.

7.2.3 Aggregation of search results

After the search process is terminated, peer p_I merges query answers from different peers into a single query answer. First, the relevance score $score(q, d)$ is computed for every retrieved document d as it was discussed in Section 4.3. Note that the number of documents (N) is computed as a sum of all the $numDocs_p$, and document frequency (n) is computed as a sum of all the $docFreq_p(A)$. Second, the document score $score(q, d)$ is combined with the score $score(q, n)$ of the node n to which the document is classified in order to compute the final score of the document:

$$score'(q, d) = score(q, n) + score(q, d),$$

Finally, documents are ordered according to the relevance score and presented to the user in the decreasing order of relevance.

7.3 Semantic Link Discovery

When a new peer joins the network, there are no semantic links connecting the nodes in a classification of this peer with the nodes in classifications of other peers. In fact, this is not the only scenario in which links can be missing. Sometimes users become interested in new topics and therefore they e.g. can create new nodes or issue queries that are not related to any of nodes in their classifications. In the following we discuss how new semantic links can be discovered in these and other similar situations.

If two classifications which need to be connected are known in advance, then semantic links between these classifications can be computed by using semantic matching (*S-Match*) [38]. When the relevant classifications are not known, one way of computing semantic links is to run S-Match between the given classification and all the classifications of other peers. The problem with this approach is that the number of peers in the network can be huge and, therefore, running S-Match for all the possible combinations of classifications can become unfeasible.

In order to allow for an efficient discovery of semantic links, we propose to use *P2P Concept Search* approach (see Chapter 6). In this chapter, P2P Concept Search is used in order to index and retrieve complex concepts at nodes $C^{n'}$. In this case, the query answer $QA(C, \mathcal{T}_{p2p})$ (see Equation 6.1), produced by P2P Concept Search for complex concept C, contains a set of nodes n' in which complex concepts $C^{n'}$ are more specific than C:

$$QA(C, \mathcal{T}_{p2p}) = \{n' \mid \mathcal{T}_{p2p} \models C^{n'} \sqsubseteq C\}$$

If the user wants to discover semantic links for a node n or a query q, first, the query answer $A(C, \mathcal{T}_{p2p})$ is computed using the complex concept C^n or C^q accordingly. The system then returns the ordered list of possibly relevant nodes to the user. Note that if no exact matches are found, partial matches are returned, i.e., when not all atomic concepts $A \in C^n$ are used.

Finally, semantic links are created for those nodes which are selected by the user. Links created by users are indexed in the DHT by id's of target nodes. Note that these links can be used in the computation of node score in Equation 7.4. After semantic links are discovered for a node (or a set of nodes), S-Match can be run between the user's classification and some of the classifications which are connected by the links.

One of the main problems of DHT based IR is that the storage and bandwidth required for inverted lists, e.g., during inverted list intersection, can exceed the maximum allowed limits (see Section 3.2). In our approach P2P Concept Search is used only to retrieve nodes and not documents. Note that the number of nodes and size of their labels are usually smaller then the number and size of documents classified to these nodes. As a result, less storage space is needed for storing the inverted lists. Moreover, a node has only one complex concept and, therefore, intersection of inverted indices is not required which reduces a bandwidth consumption.

7.4 Summary

In this chapter, we showed how links in classification hierarchies together with semantic links which codify the mappings existing among nodes from classifications of different peers can be used to define a semantic overlay network which can cover any number of peers (e.g., in the Web). In the semantic overlay, peers with similar content are connected to each other by the means of semantic links. Differently from [22], a global classification is not required and users are free to create their own classification hierarchies. We have also shown how the semantic overlay network can be flooded and used to perform semantic search on links. Finally different ways for performing link discovery were discussed.

Part IV

Evaluation

Chapter 8

Automatic Data-Set Generation

The conference series like TREC provide different *manually* built data-sets for evaluation of search systems performance on various IR tasks. For instance, the *Ad Hoc* task is used for evaluation of free-text retrieval and it examines the performance of systems where the set of documents is fixed and the query set is not known before the experiment. In the rest of this chapter, we propose an approach for *automatic* generation of IR data-sets based on search engines query logs and data from human-edited web directories. Material presented in this chapter has been published in [47].

8.1 Data-Set Generation

In order to evaluate the efficiency of an *IR* system, we need a data-set which consists at least of the following three components:

Documents (D): Traditionally, documents are represented as Natural Language (NL) texts which vary in size, use different vocabularies, and are about different subject matters. Since most of the real *IR* systems need to deal with large document collections, the set of documents in the data-set should be also big enough. Otherwise, the

results obtained on the data-set can not be considered as a good approximation of the real performance of the evaluated system.

Queries (Q): Queries are short statements of user information needs. In fact, the average size of queries which are submitted to the current search engines is less than three words. Such short queries can be ambiguous. In order to be able to evaluate the quality of the results returned by an *IR* system, the data-set should provide an unambiguous description for these queries. For instance, each query in the ad-hoc TREC document collection[1] is assigned a description, i.e., one sentence which describes a topic area, and a narrative, i.e., a concise description of what makes a document relevant to the query.

Relevance judgments (R): A relevance judgment is a query-document pair where the relevance of the document to the query is specified. For instance, in TREC, the binary relevance judgment is used, i.e., either a document is relevant to the query or it is not. The following rule is used by TREC assessors to evaluate the relevancy of a document to the query. If any information contained in a document can be used to write a report about subject of the query, then the document should be marked as relevant. In ideal case, a set of relevancy judgments should be complete and correct. In reality, the size of document collections make it infeasible to produce the complete set of relevance judgments, and, therefore, some approximation of the relevancy judgments set is used instead. For example, the pooling methodology is used in TREC [96] to provide such approximation.

In this chapter, we propose an approach for automatic generation of data-sets by using search engines query logs and data from human-edited web directories. We use the AOL query log [66], which consists of over 20,000,000

[1] `http://trec.nist.gov/data/test_coll.html`

queries made by over 500,000 AOL users during a three-month period. As a web directory we use the *Open Directory Project*[2], (*ODP*) also known as *DMoz*. *DMoz* is a multilingual open content directory of World Wide Web links that is constructed and maintained by a community of more than 80,000 volunteer editors. The *DMoz* directory contains over 590,000 multilingual categories organized into a hierarchy and over 4,500,000 web-sites classified to these categories. The meaning of each category is defined by its position in the hierarchy. For instance, category *Languages*, which can be reached by a path *Top/Computers/Programming/Languages/* represents a set of web-sites about programming languages and directly related topics. Moreover, all the sub-categories of this category also need to be related to *programming languages*. For instance, category *Java* with the path *Top/Computers/Programming/Languages/ Java/* is about programming language Java. Each web-site, in *Dmoz*, is represented by an URL, a title, and a short description of its content. Web-sites are classified to categories according to the *get-specific rule* (see Section 5.1), i.e., the category which describe the content of the web-site in the most specific way should be chosen. In the following, we discuss how *AOL* query log and *DMoz* web directory can be used for automatic generation of data-sets.

The documents, in the data-set, are created by using web-sites classified in *DMoz*. First, we collect all the URLs of web-sites classified in DMoz. Note, that we excluded from consideration all the web sites classified in *Adult*, *World*, *Regional* and *Kids_and_Teens* sub-trees. *Adult* sub-tree is excluded because it can contain web-sites with inappropriate adult content, *World* sub-tree is excluded because it contain web-sites with non-English content, and both *Kids_and_Teens* and *Regional* sub-trees are excluded because they have guidelines which are different from those for the rest of the directory. Second, for every URL, we fetch a single web-page pointed

[2] `http://www.dmoz.org/`

by the URL. Third, for every web-page we extract out-links, i.e., URLs which appear in the web-page together with their anchor, and, if there is an out-link with the phrase *about us* (or *about me*), we fetch the web-page corresponding to this URL. All the markup is eliminated from first and *about us* web pages. The fetching of web-site contents and elimination of the markup is implemented by using Nutch[3].

In this chapter, as a document set we used only those web-sites which have 'about us' web-pages. We use *AboutUs* as a name for the data-set. Every document in the *AboutUs* data-set consists of three textual fields, which describe what the corresponding web-site is about:

Description In DMoz, for each web-site, there is a short description, written by a DMoz editor, which describes what the web-site is about from her point of view. "*The description gives specific information about the content and/or subject matter of the site. It should be informative and concise, usually no longer than one or two lines. The basic formula for a good description is: Description = Subject + Content. ... End users should be able to determine relevancy without having to visit a site.*"[4]

First page First page is the first (and probably the last) think that user see when she visits the web-site. So, the first page should usually give a good idea about web-site content. We see the first page as a description of what the web site is about from the point of view of a web-site visitors.

'About us' Web-site's about page describes what the web site is about from the point of view of web-site authors.

[3] `http://lucene.apache.org/nutch/`

[4] `http://www.dmoz.org/guidelines/describing.html#descriptions`

Note, that other web-pages, which can be reached from the first page, can also be used to describe the topic and the content of the web-site. The problem is that it is hard to distinguish between these pages and the ones which are completely unrelated.

In order to generate a query set, we first, collect all the unique queries from AOL query log. One word queries, queries which contain punctuation, special symbols, or boolean operators (e.g., '+', and '?'), and queries which contain the words shorter than 3 letters are filtered out. Second, for every query, we search for a set **C** of DMoz categories with titles consisting only of the query words (we used exact matching of lowercased words). For example, for query *africa scuba diving* we find categories *Africa* and *Scuba_Diving*. Third, for every category in **C**, we check if its path to the root contains a combination of categories (which are also in **C**), which all together contain all and only query words. For example, the path to the root *Top/Recreation/Outdoors/Scuba_Diving/Regional/Africa* has two categories *Scuba_Diving* and *Africa* with all and only words from the given query. In order to have queries with only one possible interpretation, we filtered out all the queries which matched more than on paths to the root. In Table 8.1, we show some examples of query-category pairs which we obtained as a result of the described above process. Notice, that many categories in DMoz are assigned descriptions. These descriptions, similarly to the query descriptions in TREC collections, can be used to describe the meaning of the query in the corresponding query-category pair.

In order to generate a set of relevance judgments, we used a mapping from queries to categories obtained as described above and also a mapping from categories to the documents classified to these categories by DMoz editors. For every category, we collect all the documents classified to this category plus all the documents classified to more specific categories. All the documents collected for a category are considered to be relevant to

Table 8.1: Query-category pairs

AOL Query	DMoz Category
africa scuba diving	Top/Recreation/Outdoors/Scuba_Diving/Regional/Africa
analytical chemistry	Top/Science/Chemistry/Analytical
breast cancer organizations	Top/Health/Conditions_and_Diseases/Cancer/Breast/Organizations
business awards	Top/Business/Consumer_Goods_and_Services/Awards
home based business opportunities	Top/Business/Opportunities/Home_Based
homebrewing beer	Top/Recreation/Food/Drink/Beer/Homebrewing
knowledge management	Top/Reference/Knowledge_Management
laser toner	Top/Computers/Hardware/Peripherals/Printers/Supplies/Laser_Toner
lions clubs international	Top/Society/Organizations/Service_Clubs/Lions_Clubs_International
luxury jewelry	Top/Shopping/Jewelry/Watches/Luxury
nuclear magnetic resonance	Top/Science/Chemistry/Nuclear_Magnetic_Resonance
photography education	Top/Arts/Photography/Education
rehabilitation medicine	Top/Health/Medicine/Medical_Specialties/Rehabilitation_Medicine
rugby football union	Top/Sports/Football/Rugby_Union
shih tzu breeders	Top/Recreation/Pets/Dogs/Breeds/Toy_Group/Shih_Tzu/Breeders
small business accounting software	Top/Computers/Software/Accounting/Small_Business
solar energy business	Top/Business/Energy/Renewable/Solar
travel around the world	Top/Recreation/Travel/Travelogues/Around_the_World
united states adoption	Top/Home/Family/Adoption/Wish_to_Adopt/Regional/United_States
yellow pages directories	Top/Reference/Directories/Address_and_Phone_Numbers/Yellow_Pages

the query in the corresponding query-category pair. Here, the intuition is that, since documents in *DMoz* are sub-categorized and organized by topics[5], all the documents classified in the sub-tree should be relevant to all the categories on the path to the root, including those categories which are matched by the query words. Trivial query-document matches, i.e., the ones where documents include query as an exact phrase were excluded from the data-set together with corresponding documents. For example, for a query "west highland white terrier", document "The West Highland White Terrier is a small terrier" is considered trivial, because any syntactic or semantic technique can trivially find this document. Moreover, we pruned all the queries which have less than 10 relevant results. The statistics of the resulting *AboutUs* data-set is summarized in Table 8.2[6]. Notice that the generated set of relevance judgments is correct and complete, in the

[5] `http://www.dmoz.org/guidelines/subcategories.html`

[6] White space is used as an indication of a separation between words

Table 8.2: *AboutUs* data-set statistics

Statistics category	Value
Documents	100,807
Queries	330
Relevance judgments	8,704
Query length (words), avg.	2.4
Description length (words), avg.	16.0
First page length (words), avg.	485.4
'About Us' page length (words), avg.	473.3

case, when (i) editors do not do mistakes and do not miss relevant documents, and (ii) the document description is rich enough to judge about the relevance of this document to the query. According to *DMoz* guidelines: "*An effective editor will search and/or browse through the ODP in areas inside and outside his or her top level category to find areas of potential duplication*"[5]. Assuming that most of editors are "effective editors", the set of relevance judgments (obtained by the described above approach) should be a good approximation of the ideal (i.e., complete and correct) set of relevance judgments. The impact of the richness of the document descriptions on the performance of search techniques is studied in the following section.

8.2 Summary

In this chapter, we presented an approach for automatic generation of the data-sets for evaluation of the *ad hoc* retrieval task. The generation of the data-sets is done using search engines query logs and data from human-edited web directories.

Chapter 9

Evaluation Results

This chapter provides evaluation results for the algorithms described in Chapters 4, 5, 6, and 7. Namely, evaluation results for *C-Search* algorithm are discussed in Section 9.1. Evaluation results for *Get-Specific* document classification algorithm are discussed in Section 9.2. Evaluation results for *P2P C-Search* algorithm are discussed in Section 9.3 and *Semantic Flooding* algorithm is evaluated in Section 9.4.

9.1 Concept Search

In order to evaluate the *C-Search* approaches (see Chapter 4), we built six IR systems. One system is an instantiation of syntactic search and is build on top of Lucene. Standard tokenization and English Snowball stemmer were used for document and query preprocessing. The AND operator was used as a default boolean operator in a query. The Lucene default implementation of the cosine similarity from the vector space model was used for relevancy ranking[1]. Other five systems are semantics enabled versions of Lucene, implemented following the approaches described in Sections 4.4.1-4.4.5. WordNet 2.1 was used as a lexical database in all these systems. GATE [25] was used in order to locate descriptive phrases (see

[1] `http://lucene.apache.org/java/2_4_0/api/org/apache/lucene/search/Similarity.html`

Section 4.1). Relevancy ranking, in these systems, was implemented by modifying Lucene default score function[1] as it was described in Section 4.3.

9.1.1 Quality Evaluation: TREC Data-Set

In this section, we compared the quality of results returned by Lucene and by *C-Search*. *Approach 3* was used for this evaluation but, given that the approaches 1, 2 and 4 implement the same algorithm, the results returned by these approaches should be comparable. As a data-set for our experiments, we used the TREC ad-hoc document collection[2] (disks 4 and 5 minus the Congressional Record documents) and three query sets: TREC6 (topics 301-350), TREC7 (topics 351-400) and TREC8 (topics 401-450). Only the title for each topic was used as a query. The whole data-set consists of 523,822 documents and 150 queries. In the evaluation we used the standard IR measures and in particular the mean average precision (MAP) and precision at K (P@K), where K was set to 5, 10, and 15. The average precision for a query is the mean of the precision obtained after each relevant document is retrieved (using 0 as the precision for not retrieved documents which are relevant). MAP is the mean of the average precisions for all the queries in the test collection. P@K is the percentage of relevant documents among the top K ranked documents. MAP is used to evaluate the overall accuracy of IR system, while P@K is used to evaluate the utility of IR system for users who only see the top K results.

First, in Table 9.1, we report the evaluation results for the two systems and further, in Figure 9.1 we provide recall-precision graphs, i.e., we plot precision as a function of recall, for these systems. The experiments show that, on TREC ad-hoc data sets, *C-Search* performs better than the purely syntactic search, which supports the underlying assumption of our approach. In particular, from Table 9.1 we observe that *C-Search* improves

[2] `http://trec.nist.gov/data/test_coll.html`

Table 9.1: Evaluation results

TREC6 (301-350)				
	MAP	P@5	P@10	P@15
Lucene	0.1361	0.3200	0.2960	0.2573
C-Search(Lucene)	0.1711(+25.7%)	0.3920(+22.5%)	0.3480(+17.6%)	0.3000(+16.6%)
TREC7 (351-400)				
	MAP	P@5	P@10	P@15
Lucene	0.1138	0.3560	0.3280	0.3000
C-Search(Lucene)	0.1375(+20.8%)	0.4200(+18.0%)	0.3680(+12.2%)	0.3427(+14.2%)
TREC8 (401-450)				
	MAP	P@5	P@10	P@15
Lucene	0.1689	0.4320	0.4000	0.3573
C-Search(Lucene)	0.2070(+22.6%)	0.4760(+10.2%)	0.4280(+7.0%)	0.4013(+12.3%)

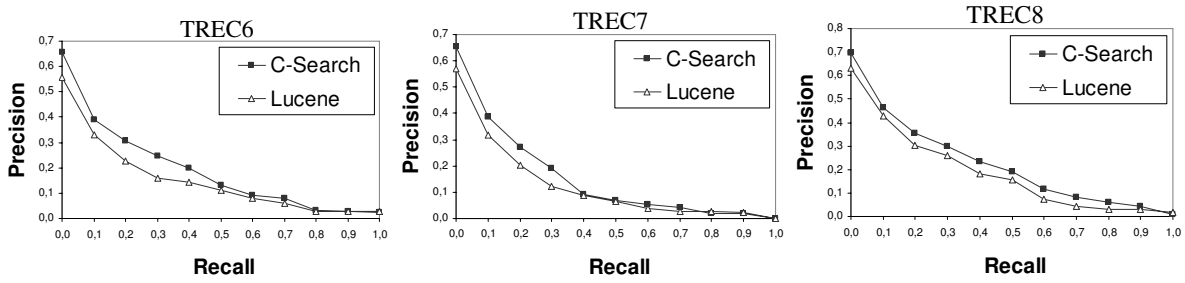


Figure 9.1: Recall-Precision Graphs

precision P@K for all K in all three TREC data sets. This is coherent with the intuition that semantics improve on precision. Notice that it means that we are able to show to the users more relevant documents at the top of the list. From Figure 9.1 we observe that the recall-precision graphs for *C-Search* are above those for Lucene, which means that the improvement in precision achieved by *C-Search* does not decrease recall.

9.1.2 Quality Evaluation: Document Size

In this section, we study how a size of a web-site description, which can be used as a rough indicator of the amount of available information about the web-site, and the level of details in the description, can affect the performance of search techniques. Three data-sets were generated based on the *AboutUs* data-set (see Section 8.1). These data-sets represent differen lev-

	descr			**descr+fp**			**descr+fp+ap**		
	MAP	P@5	P@10	MAP	P@5	P@10	MAP	P@5	P@10
Lucene	0.0200	0.0879	0.0558	0.1008	0.2255	0.1945	0.1349	0.2473	0.2236
C-Search(Lucene)	0.0359	0.1230	0.0924	0.1411	0.2345	0.2182	0.1798	0.2685	0.2524
Improvement	+79.5%	+39.9%	+65.6%	+40.0%	+4.0%	+12.2%	+33.3%	+8.6%	+12.9%

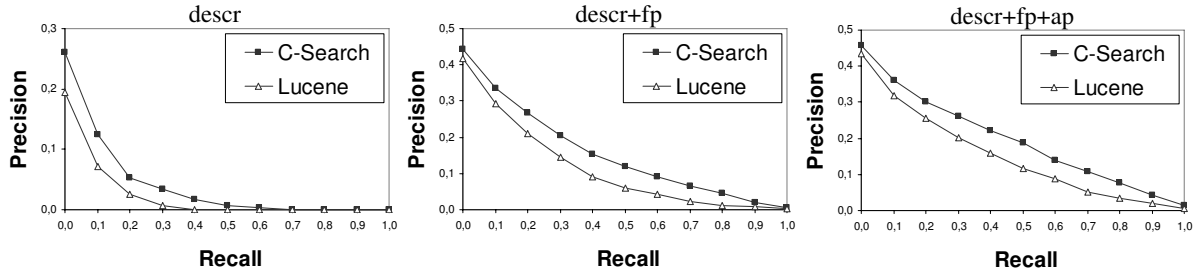


Figure 9.2: Evaluation results: Document Size

els of details in document description. The first data-set (**descr**) consists only of short descriptions, created by DMoz editors, which briefly describe the web-site. In the second data-set (**descr+fp**), every document is composed from the description and the text from the first page of the web-site. The third data-set (**descr+fp+ap**) consists of the documents, which are composed from description, first and 'about us' web-pages. Actually, **descr+fp+ap** represents the complete *AboutUs* data-set.

We evaluated the performance of Lucene and *C-Search* on all three data-sets. The evaluation results are reported in Figure 9.2. The experiments show, that, the bigger is the document description, the easier is the search task for both Lucene and *C-Search*. After manual inspection of the results, we concluded that the main reason for this is the increase in the quality of the data-set. If a document description is only a short summary of a web-site (as it is the case in the **descr** data-set), it may not be relevant to a query created for a category in which the web-site is classified. For instance, let us consider the following document description: *Links to auto reviews and articles.* The description is created for the web-

site classified to category *New*[3] and, therefore, this document description can be associated with the query *purchasing new automobiles*, but, as we can see, this description contains no information relevant to purchasing of something new. If, in addition to the description, we consider also the first page (as in the **descr+fp** data-set), and 'About Us' page (as in the **descr+fp+ap** data-set) then the web-site description become more complete and the search techniques improve their results. Note, however, that data collected from web-sites can be very noisy, because usually there are many advertisements on web-sites and/or because web-site administrators use different search engine optimization (SEO) techniques, such as adding popular keywords to their web-pages in order to improve the find-ability of their web-sites. In general, it can cause decrease in precision. As we can observe from Figure 9.2, the incompleteness of the document descriptions and the noisiness of web-pages are not playing decisive role if we want to conduct comparative evaluation of different search techniques. For example, C-Search performs better than Lucene on all three data-sets.

9.1.3 Quality Evaluation: Semantic Heterogeneity

In the context of IR, semantic heterogeneity refers to a phenomenon, when a person submitting a search query and authors of documents have no agreement about how to represent the same or related objects. For instance, it can lead to the situation, when words which are used to describe the object in a query are different from those words which are used to describe the same object in the document description. In this section, we study how the semantic heterogeneity problem can affect the performance of search techniques.

We create three data-sets: **descr+fp+ap_25**, **descr+fp+ap_10**, and **descr+fp+ap_0**, which are based on *AboutUs* data-set (**descr+fp+ap**).

[3] http://www.dmoz.org/Home/Consumer_Information/Automobiles/Purchasing/New/

	descr+fp+ap_25			**descr+fp+ap_10**			**descr+fp+ap_0**		
	MAP	P@5	P@10	MAP	P@5	P@10	MAP	P@5	P@10
Lucene	0.1098	0.2188	0.1724	0.0466	0.1497	0.1064	0.0000	0.0000	0.0000
C-Search(Lucene)	0.1543	0.2400	0.2079	0.1178	0.1824	0.1496	0.0604	0.0804	0.0707
Improvement	+40.5%	+9.7%	+20.6%	+152.8%	+21.8%	+40.7%	-	-	-

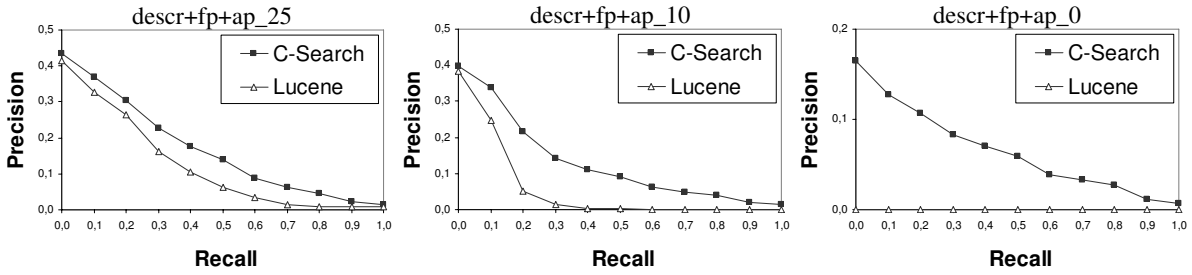


Figure 9.3: Evaluation results: Semantic Heterogeneity

The number X, which appears at the end of the data-set name **descr+fp+ap_X**, represents the percentage of relevant documents which can have all the words from the corresponding query. The data-sets were created by excluding all the documents and corresponding relevance judgments which were above the specified limit. Notice, that the bigger is X, the higher is the level of semantic heterogeneity, where the **descr+fp+ap_0** data-set represents the extreme case when syntactic search is not possible.

The performance of Lucene and *C-Search* was evaluated on these data-sets. The evaluation results are reported in Figure 9.3. From Figure 9.3, we observe that improvements, achieved by *C-Search*, starts being significant when the heterogeneity is high (i.e., when the number X is small).

In order to compare the level of semantic heterogeneity in the generated data-sets with those in standard *IR* data-sets, we took three TREC data-sets: TREC6 (topics 301-350), TREC7 (topics 351-400), and TREC8 (topics 401-450). The average number (and the average percentage) of relevant documents which have all the query words is computed for these data-sets (see Table 9.2). As we can see from Table 9.2, in TREC data-sets, more than 20 relevant documents in average can be retrieved by syntac-

Table 9.2: Semantic heterogeneity in TREC ad-hoc data-sets

Data-set	Average number of relevant documents which contain all the query words	Average percentage of relevant documents which contain all the query words
TREC6	23.9	27.32 %
TREC7	24.2	29.67 %
TREC8	34.7	40.98 %

tic matching of document and query words. These documents in average amount to more than 25% of all the relevant documents. The level of semantic heterogeneity problem in TREC data-sets is rather low to show the advantages of semantic techniques (especially when retrieval of top-k results is considered).

9.1.4 Performance Evaluation

In this section, we compared an index size and a search time of different versions of *C-Search*. TREC was used as a document collection. Three query sets, with queries consisting of: (i) 1 word, (ii) 2 words, and (iii) 3 words were generated by randomly selecting a set of 1000 queries from the AOL query log [66] for each query set. Queries which contain punctuation, special symbols, or boolean operators (e.g., '+', ' ', and '?'); queries which contain the words shorter than 3 letters; and queries which didn't have any results were filtered out. All the experiments described in this section were run on a machine with the following parameters:

- CPU : Intel(R) Core(TM)2 Duo T7500 @2.20GHz
- RAM : 3GB
- HD : 250GB @ 5400 RPM
- OS : Windows XP (SP3)

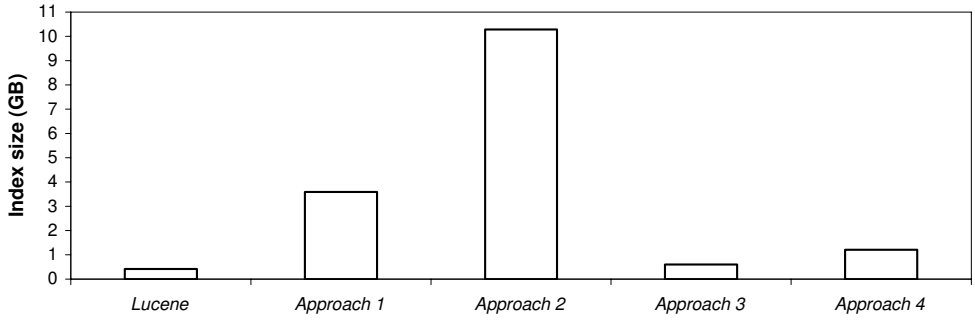


Figure 9.4: Size of the inverted index

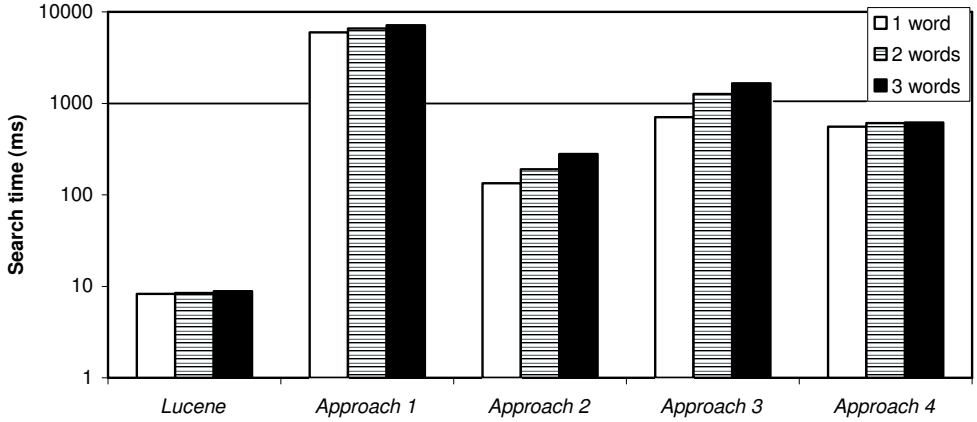


Figure 9.5: Search time

In Figure 9.4, we report on the size of the inverted indexes created by *Lucene* and by *C-Search* (approaches 1-4). In Figure 9.5, we report an average search time per query in milliseconds (ms)[4]. As we can observe from Figure 9.5, *Approach 2* is the the fastest among of *C-Search* approachs. It can provide answers in less than a second, namely in a time which is acceptable for the user to wait. The main reason why *Approach 2* is still much slower than *Lucene* is that, in *C-Search*, we need to analyze positions of atomic concepts and not just the number of their occurrences. Note that the number of positions which need to be analyzed can be much bigger than the number of relevant documents (especially for a very general query concept). The large size of the index in Approach 2 (see Figure 9.4)

[4]Every experiment was run 5 times and the average result was reported.

is also due to the large amount of positions which need to be stored (see Section 4.4.2 for details). On the contrary, in *Approach 3*, the size of the index is the smallest among all the other *C-Search* approaches and the average search time is within a few seconds. *Approach 4* improves the search time of *Approach 3* at the cost of doubling its index size.

9.1.5 Quality vs. Performance

In this section, we studied the influence of the following two parameters on the quality of results and performance of *Approach 5* (see Section 4.4.5): (i) s - a maximum number of senses which can be assigned to a word in a query after the word sense filtering step; and (ii) *dist* - a maximum allowed distance between atomic concepts in a query and a document. As a data-set, we used the data-set described in Section 9.1.1.

Figure 9.6 shows the influence of the parameter s on the search time (represented as bars) and MAP (represented as broken lines) for *Approach 5* on the three query sets: TREC6, TREC7, and TREC8. As we can see from Figure 9.6, the quality of results (measured by MAP) returned by the approximated version of *C-Search* is not decreasing much if we use s equal to or bigger than three. At the same time, the search time decreases substantially, namely, if s is reduced from 7 to 3, the search time becomes three times smaller on the TREC7 query set. In Figure 9.7, we show how the search time and MAP for *Approach 5* are influenced by the parameter *dist* (where s was set to 3). As we can see from Figure 9.7, the MAP remains almost constant if we keep *dist* equal or bigger than three. If *dist* is set to 3, the search time is decreased around two times, with respect to the case when *dist* is not limited. In total, by using $s = 3$ and $dist = 3$, *Approach 5* can perform 6 times faster than *Approach 3*, while having almost no decrease in the quality of results measured by MAP.

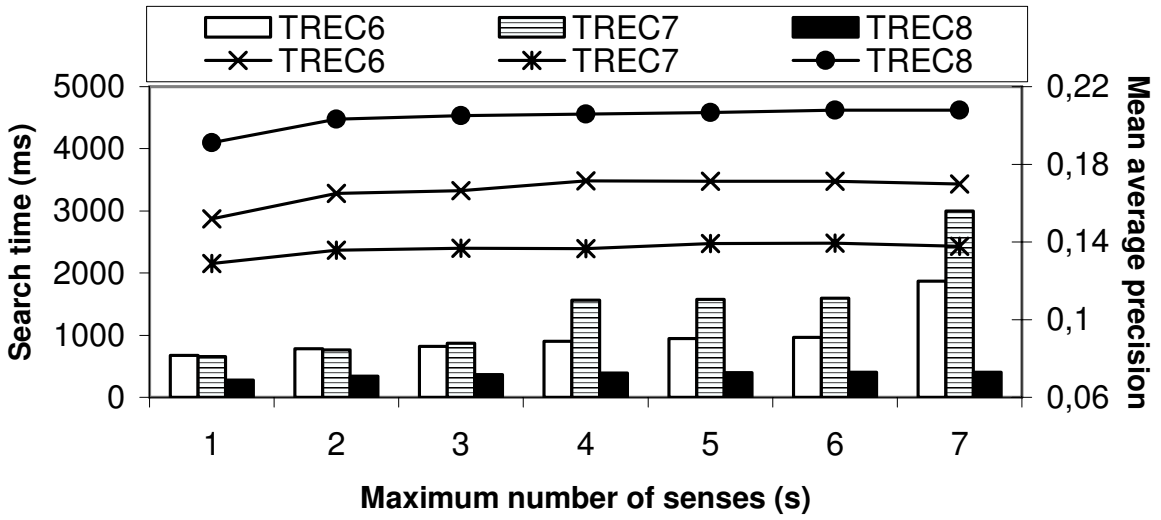


Figure 9.6: Influence of a max number of senses for a word on a search time and MAP

9.2 Document Classification

In order to evaluate the get-specific document classification algorithm (see Chapter 5), we selected four subtrees from the DMoz web directory, converted them to FCs, extracted concepts from the populated documents, and automatically (re)classified the documents into the FCs by using the get-specific algorithm. Document concepts were extracted by computing the conjunction of the formulas corresponding to the first 10 most frequent words appearing in the documents (excluding stop words). We used WordNet [60] for finding word senses and their relations, and we used S-Match [38] for computing Equation 5.7. Parameter k for tradeoff computation was set to 2.

In the evaluation we employ standard IR measures such as micro- and macro-averaged precision, recall, and F1 [82]. In Table 9.3 we report data-set statistics and evaluation results for each of the four data-sets. We performed a detailed analysis of the "Languages" data-set results (see Figure 9.8). In Figure 9.8a we show how precision and recall are distributed among nodes. Figure 9.8b shows how far (in terms of the number of edges)

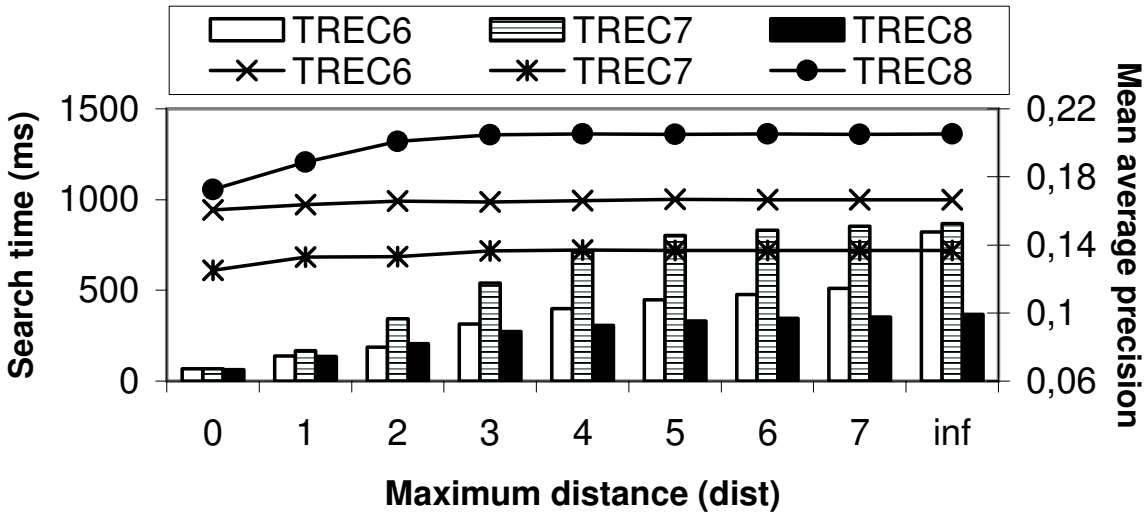


Figure 9.7: Influence of a max distance between concepts on a search time and MAP

Table 9.3: Data-set statistics and evaluation results

Data-set	Nodes	Docs	Max. subtree depth	Mi-Pr	Mi-Re	Mi-F1	Ma-Pr	Ma-Re	Ma-F1
Photography[a]	27	871	4	0.2218	0.1871	0.2029	0.2046	0.1165	0.1485
Beverages[b]	38	1456	5	0.4037	0.4938	0.4442	0.3848	0.3551	0.3693
Mammals[c]	88	574	5	0.3145	0.3014	0.3078	0.3854	0.2677	0.3159
Languages[d]	157	1217	6	0.4117	0.4503	0.4301	0.4366	0.4187	0.4275

[a] http://dmoz.org/Shopping/Photography/.

[b] http://dmoz.org/Shopping/Food/Beverages/.

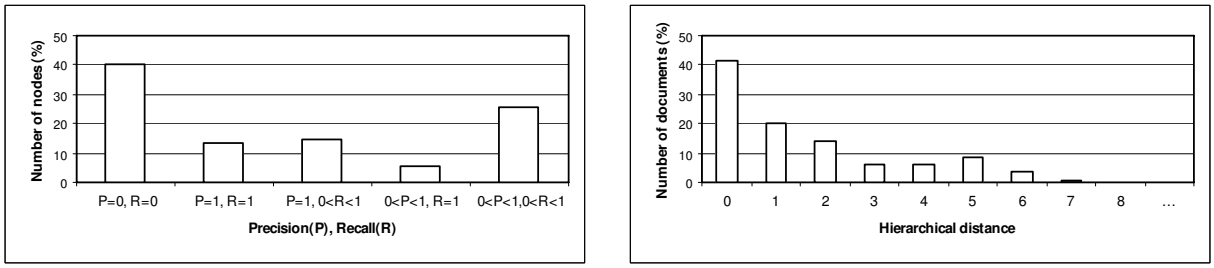
[c] http://dmoz.org/Health/Animal/Mammals/.

[d] http://dmoz.org/Science/Social_Sciences/Linguistics/Languages/Natural/Indo-European/.

an automatically classified document is from the node where it was actually classified in DMoz.

From Figure 9.8a we observe that about 40% of nodes in the "Languages" data-set have precision and recall equal to 0[5]. After manual inspection of the results, we concluded that this problem is caused by lack of background knowledge. For instance, 8 documents about Slovenian language were misclassified because there was no WordNet synset "Slovenian" defined as "the Slavic language spoken in Slovenia" and a hypernym re-

[5] Precision for nodes with no documents was counted as 0.



(a) (b)

Figure 9.8: Analysis of the "Languages" data-set results

lation of it with synset "Slavic language". Figure 9.8b shows that about 20% of documents are classified in one edge distance from the node where they were originally populated, whereas 89% of them were classified one node higher on the path to the root. Note that this still allows it to find a document of interest by browsing the classification hierarchy.

9.3 P2P C-Search

In order to evaluate *P2P C-Search* approach, we developed real implementations of the following two prototypes: P2P Syntactic Search (see Section 3.2) which is based on Lucene, and P2P C-Search (see Chapter 6) built on top of *C-Search* (see Section 4.4.3)[6]. The performances of P2P Syntactic Search and P2P C-Search were compared. Experiments with the real implementation could not be performed on thousands of nodes due to physical limitations. Hence, a custom simulator was developed by reusing the real implementation. For validation of the simulator, the real implementations of P2P Syntactic Search and P2P C-Search were tested on a cluster of 47 heterogeneous nodes. The same queries were performed on the simulator and the results were found exactly same as that in a real

[6]To validate that there are no optimizations which can affect the fair comparison between syntactic and semantic approaches we compared the results of syntactic approach with the results of semantic approach when the background knowledge is empty. No significant difference was found.

Table 9.4: Evaluation results: Syntactic vs. Semantic

TREC8 (401-450)		
	P2P Syn. Search	P2P C-Search
MAP	0.1648	0.1884(+14.3%)
P@5	0.4040	0.4440(+9.9%)
P@10	0.3860	0.4200(+8.8%)
P@15	0.3733	0.3907(+4.7%)

network setting.

As a data-set for this experiment, we used the TREC ad-hoc document collection and the query set from TREC8 (topics 401-450). The results of the evaluation are shown in Table 9.4. The experiments show that, on TREC ad-hoc data set the results achieved by *P2P C-Search* are better than those achieved by P2P syntactic search. Note that the results achieved by distributed approaches are comparable with the results of the centralized versions of Lucene and *C-Search* (see Section 9.1.1), i.e., quality is not lost much due to distribution.

The average number of postings (document id and additional information related e.g. to score of the document) transferred per query (network overhead) was 49710.74 for P2P syntactic search and 94696.44 for P2P C-Search. Thus the network overhead of P2P C-Search is 1.9 times that of syntactic search. But, the average length of intermediate posting lists transferred for P2P C-Search is only 37.48% as that of syntactic search even though the cumulative size is bigger. Thus by incorporating the optimizations proposed in Section 6.2 (i.e. pre-compute addresses of peers responsible for more specific and getting respective postings in parallel), the response time for semantic search could be reduced compared to that of syntactic search. But the current basic prototype, doesn't include sophisticated optimizations and hence the search time comparisons are not made.

The number of postings transferred per query was taken as the measurement of network bandwidth consumption as they form the majority of network traffic for search (DHT lookup cost is comparatively negligible). Also, different optimizations like Bloom filters can be used to improve the transfer bandwidth for both syntactic and concept search.

9.4 Semantic Flooding

In order to evaluate Semantic Flooding approach (see Chapter 7), we conducted a set of simulation experiments, where the results of distributed *Semantic Flooding* algorithm and of centralized *C-Search* were compared. The key intuition here is to see how much we lost, in terms of the number of documents which are retrieved by a centralized search approach and which are missing from the results of a distributed search approach. In the evaluation we measured the accuracy of search results depending on the number of visited peers, where the accuracy is defined as follows:

$$Accuracy = \frac{|R_{CS} \cap R_{SF}|}{|R_{CS}|} * 100\%,$$

where R_{CS} are results returned by *C-Search* and R_{SF} are results returned by *Semantic Flooding*. Only the first 10 results were considered.

Four data-sets were generated by using data from the *DMoz* web directory. The data-sets consist of 10, 100, 1000, and 10000 peers, where a classification of each peer was generated by extracting a part of the *DMoz* classification. The generation of a classification was performed as follows. First, N_s sub-trees were randomly selected in *DMoz*, where N_s is a randomly selected number in a range [10..20]. We used only those sub-trees in *DMoz* which are rooted by nodes at the second level in the *DMoz* classification (e.g., *Top/Arts/Animation/* and *Top/Health/Alternative*). Second, N_p paths from leaf nodes to the root node were randomly selected

from these sub-trees ($N_p \in [90..110]$). Third, a random number of nodes were selected from every path. Then, a set of N_d documents was randomly selected for each node ($N_d \in [10..30]$). Documents in the data-set were created by concatenating titles and descriptions of web-sites. Finally, the classification was created from all the nodes, and the documents were classified to these nodes. On average, classification of each peer have 159 nodes and 768 documents. For each data-set, a *C-Search* index I_{CS} was created. All the documents in the data-set were indexed in I_{CS}. Indexes of each single peers were created by filtering I_{CS}. Note that classifications of different peers can partially overlap, where the overlap have a higher probability on the higher levels of classifications (e.g., *Top/Business/* and *Top/Games/*). In this case, there can be many peers which share some general interests (e.g., business and games), and there can be only one (or few) peers with information about a very specific topic (e.g., *Top/Games/Gambling/Sports/Racing/Horse_Racing/*).

A query set was generated by randomly selecting a set of N_q (100) queries from the AOL query log [66] for each data-set. One word queries; queries which contain punctuation, special symbols, or boolean operators (e.g., '+', and ' ?'); queries which contain the words shorter than 3 letters; and queries which had less than 10 results in I_{CS} were filtered out. For each query, we randomly selected a node n in *DMoz* classification, such that, query requests $\langle C^n, C^q \rangle$ have at least 10 relevant documents as computed by *C-Search*.

The evaluation results are reported in Figure 9.9. We compared the performance achieved by Semantic Flooding algorithm when: (i) the query request $\langle C^n, C^q \rangle$ (namely it consists of a starting node n with concept C^n and of a query q with a concept C^q) is used; (ii) the query request is $\langle \top, C^q \rangle$, namely the same as (i) but with no starting node, i.e., $C^n \equiv \top$; and (iii) the same as (ii) but the semantic similarity $SS(C^{n'}, C^q)$ is not

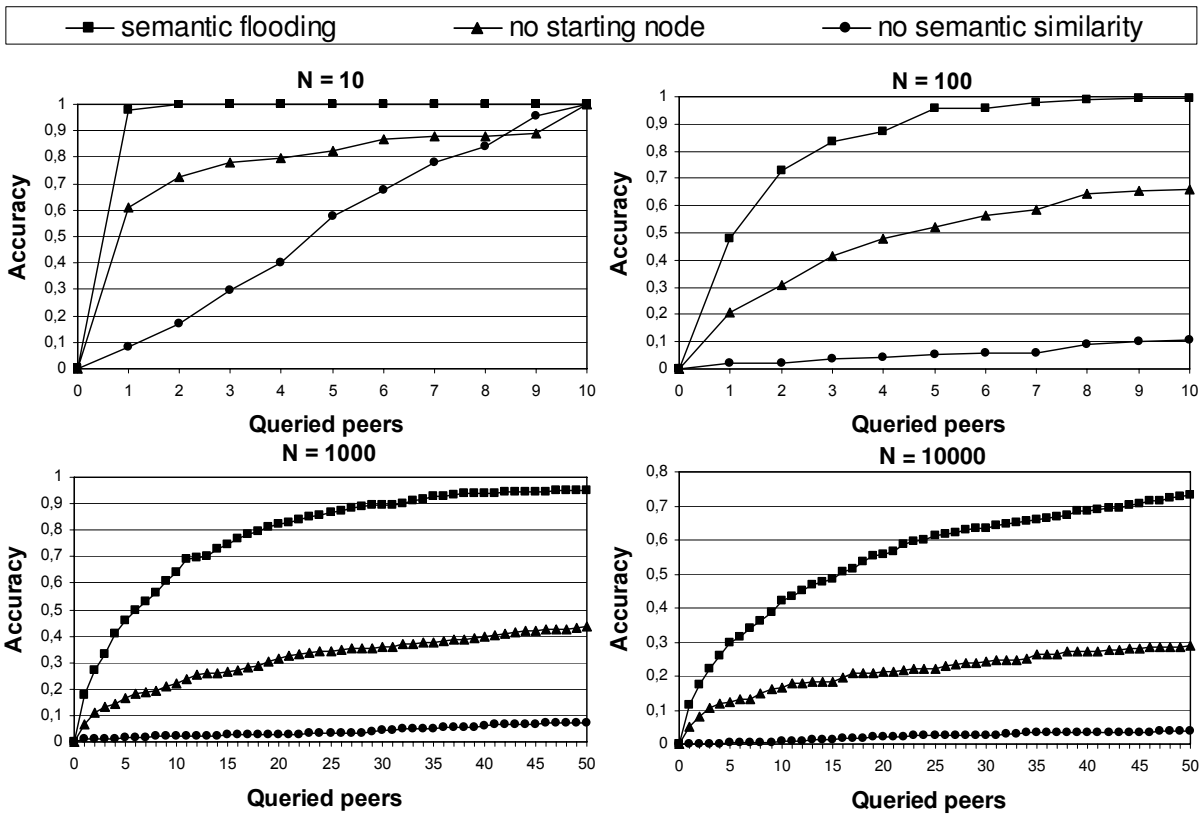


Figure 9.9: Evaluation Results

used. Note that in P2P networks of 10 and 100 peers, the total number of queried peers was set to 10, whereas in P2P networks of 1000 and 10000 peers, it was set to 50.

From Figure 9.9, we can see that, when peers are selected without using the similarity function and also without a starting node specified (see "no semantic similarity" lines in Figure 9.9), accuracy decreases very quickly with the total number of peers in the network. The situation improves when semantic similarity is used and only starting node is missing (see "no starting node" lines in Figure 9.9). When the starting node n is selected, i.e., concept C^n is provided, the accuracy of *Semantic Flooding* becomes close to the accuracy of the centralized *C-Search* approach (see "semantic flooding" lines in Figure 9.9). In fact, in the network of 10000 peers, only

50 peers need to be queried in order to achieve 70% of accuracy. Note that if we need to retrieve one relevant result (i.e., 10% of accuracy), only one peer needs to be queried.

9.5 Summary

In this chapter we have presented the evaluation results for the semantics enabled algorithms presented in previous chapters. The results are promising and demonstrate the proof of concept for the approach proposed in this thesis.

Chapter 10

Conclusions

In this thesis we presented an approach in which syntactic IR is extended with a semantics layer in order to address some of the problems of the syntactic IR approach and to improve the quality of the results returned by this approach. We concentrated on addressing the problems of (i) polysemy, (ii) synonymy, (iii) complex concepts, and (iv) related concepts. The main idea behind the proposed approach is to keep the same machinery which has made syntactic IR so successful, but to modify it so that, whenever useful, syntactic IR is substituted by semantic search, thus improving the system performance.

Several instances of the general approach applied to the problems of document classification, centralized and distributed document indexing and retrieval were presented. In Chapter 4, we described a free text document retrieval approach (*C-Search*) which is based on the semantic matching of complex concepts, where semantic matching is implemented by using (positional) inverted index. In Chapter 5, we described how the get-specific document classification algorithm, which requires that an object is classified in a category (or in a set of categories) which most specifically describes the object, can be formalized in order to fully automate it through reasoning in a propositional concept language without requiring user in-

volvement or a training data-set. In Chapter 6, we showed how *C-Search* can be extended to *P2P C-Search* approach which allows semantic search on top of distributed hash table (DHT). Differently from *C-Search*, *P2P C-Search* exploits distributed, rather than centralized, background knowledge and indices. Centralized document index is replaced by distributed index build on top of DHT. The reasoning with respect to a single background knowledge is extended to the reasoning with respect to the background knowledge distributed among all the peers in the network. In Chapter 7, we presented *Semantic Flooding* approach which exploits semantic links in the user generated classifications and also the links between the classifications of different users in order to build semantic overlay network which can be flooded to perform semantic search.

We performed an evaluation of the approaches proposed in this thesis. The reported experimental results demonstrate the proof of concept and show that proposed approaches perform as good as syntactic analogues while allowing for an improvement whenever semantics is available and can be exploited.

Bibliography

[1] Troels Andreasen, Per Anker Jensen, Jorgen Fischer Nilsson, Patrizia Paggio, Bolette Sandford Pedersen, and Hanne Erdman Thomsen. Content-based text querying with ontological descriptors. *Data & Knowledge Engineering*, 48:199–219, 2004.

[2] Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. Vector-space ranking with effective early termination. In *Proceedings of SIGIR*, pages 35–42, 2001.

[3] Vo Ngoc Anh and Alistair Moffat. Improved word-aligned binary compression for text indexing. *IEEE Transactions on Knowledge and Data Engineering*, 18:857–861, 2006.

[4] Vo Ngoc Anh and Alistair Moffat. Pruned query evaluation using pre-computed impacts. In *Proceedings of SIGIR*, pages 372–379, 2006.

[5] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2003.

[6] Mayank Bawa, Gurmeet Singh Manku, and Prabhakar Raghavan. SETS: Search enhanced by topic segmentation. In *Proceedings of SIGIR*, pages 306–313, 2003.

[7] Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. MINERVA: Collaborative P2P search. In *Proceedings of VLDB*, pages 1263–1266, 2005.

[8] Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. P2P content search: Give the web back to the people. In *Proceedings of 5th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.

[9] Michael Berry, Susan Dumais, and Gavin O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995.

[10] Ravish Bhagdev, Sam Chapman, Fabio Ciravegna, Vitaveska Lanfranchi, and Daniela Petrelli. Hybrid search: Effectively combining keywords and semantic searches. In *Proceedings of ESWC*, 2008.

[11] Bobby Bhattacharjee, Sudarshan Chawathe, Vijay Gopalakrishnan, Pete Keleher, and Bujor Silaghi. Efficient peer-to-peer searches using result-caching. In *Proceedings of the 2nd Int. Workshop on Peer-to-Peer Systems*, 2003.

[12] David Blei, Andrew Ng, and Michael Jordan. Latent Dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[13] Paolo Boldi and Sebastiano Vigna. Compressed perfect embedded skip lists for quick inverted-index lookups. In *Proceedings of SPIRE*, 2005.

[14] Alexander Borgida, Thomas Walsh, and Haym Hirsh. Towards measuring similarity in description logics. In *Proceedings of the 2005 International Workshop on Description Logics (DL2005)*, 2005.

[15] Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. Contextualizing ontologies. *Journal of Web Semantics*, 1:325–343, 2004.

[16] Alexander Budanitsky and Graeme Hirst. Evaluating WordNet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32:13–47, 2006.

[17] Stefan Büttcher and Charles Clarke. Indexing time vs. query time: Trade-offs in dynamic information retrieval systems. In *Proceedings of CIKM*, pages 317–318, 2005.

[18] Pablo Castells, Miriam Fernández, and David Vallet. An adaptation of the vector-space model for ontology-based information retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 19:261–272, 2007.

[19] Lois Mai Chan and J.S. Mitchell. *Dewey Decimal Classification: A Practical Guide*. Forest P.,U.S., 1996.

[20] Edith Cohen, Amos Fiat, and Haim Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *Proceedings of IEEE INFOCOM*, pages 1261–1271, 2003.

[21] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[22] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for P2P systems. Technical report, Stanford University, 2002.

[23] Fabio Crestani, Mounia Lalmas, Cornelis J. Van Rijsbergen, and Iain Campbell. Is this document relevant? . . . probably: A survey of prob-

abilistic models in information retrieval. *ACM Computing Surveys*, 30:528–552, 1998.

[24] W. Bruce Croft. User-specified domain knowledge for document retrieval. In *Proceedings of SIGIR*, pages 201–206, 1986.

[25] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A framework and graphical development environment for robust nlp tools and applications. In *40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

[26] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of Symposium on Operating System Design and Implementation*, 2004.

[27] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41:391–407, 1990.

[28] William A. Gale, Kenneth W. Church, and David Yarowsky. One sense per discourse. In *Proceedings of the workshop on Speech and Natural Language*, pages 233–237, 1992.

[29] Jianfeng Gao, Jian-Yun Nie, Guangyuan Wu, and Guihong Cao. Dependence language model for information retrieval. In *Proceedings of SIGIR*, pages 170–177, 2004.

[30] Steven Garcia, Hugh E. Williams, and Adam Cannane. Access-ordered indexes. In *Proceedings of Australasian Conference on Computer Science*, pages 7–14, 2004.

[31] Fausto Giunchiglia, Biswanath Dutta, and Vincenzo Maltese. Faceted lightweight ontologies. In *Conceptual Modeling: Foundations and Applications*, pages 36–51, 2009.

[32] Fausto Giunchiglia, Uladzimir Kharkevich, Alethia Hume, and Piyatat Chatvorawit. Semantic Flooding: Search over semantic links. In *Proceedings of DeSWeb 2010 workshop at ICDE*, 2010.

[33] Fausto Giunchiglia, Uladzimir Kharkevich, and Sheak Rashed Haider Noori. P2P Concept Search: Some preliminary results. In *Proceedings of SemSearch2009 workshop at WWW*, 2009.

[34] Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu. Concept Search: Semantics enabled syntactic search. In *Proceedings of SemSearch2008 workshop at ESWC*, 2008.

[35] Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu. Concept Search. In *Proceedings of ESWC*, pages 429–444, 2009.

[36] Fausto Giunchiglia, Maurizio Marchese, and Ilya Zaihrayeu. Encoding classifications into lightweight ontologies. In *Journal on Data Semantics (JoDS) VIII*, Winter 2006.

[37] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Discovering missing background knowledge in ontology matching. In *Proceedings of ECAI*, 2006.

[38] Fausto Giunchiglia, Mikalai Yatskevich, and Pavel Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics (JoDS)*, 9:1–38, 2007.

[39] Fausto Giunchiglia, Ilya Zaihrayeu, and Uladzimir Kharkevich. Formalizing the get-specific document classification algorithm. In *Proceedings of ECDL*, pages 26–37, 2007.

[40] Jiafeng Guo, Gu Xu, Hang Li, and Xueqi Cheng. A unified and discriminative model for query refinement. In *Proceedings of SIGIR*, pages 379–386, 2008.

[41] Peter Haase, Jeen Broekstra, Marc Ehrig, Maarten Menken, Peter Mika, Mariusz Olko, Michal Plechawski, Pawel Pyszlak, Björn Schnizler, Ronny Siebes, Steffen Staab, and Christoph Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proceedings of ISWC*, pages 122–136, 2004.

[42] Djoerd Hiemstra. A linguistically motivated probabilistic model of information retrieval. In *Proceedings of ECDL*, pages 569–584, 1998.

[43] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. An analysis of search-based user interaction on the semantic web. Technical Report INS-E0706, CWI, 2007.

[44] Thomas Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings of SIGIR*, pages 50–57, 1999.

[45] Dharanipragada Janakiram, Fausto Giunchiglia, Harisankar Haridas, and Uladzimir Kharkevich. Two-layered architecture for peer-to-peer concept search. Technical Report DISI-10-022, Trento University, 2010.

[46] Kalervo Järvelin, Jaana Kekäläinen, and Timo Niemi. ExpansionTool: Concept-based query expansion and construction. *Information Retrieval*, 4:231–255, 2001.

[47] Uladzimir Kharkevich. Automatic generation of a large scale semantic search evaluation data-set. In *Proceedings of ICSD*, 2009.

[48] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison Wesley, 3rd edition, 1997.

[49] Mikalai Krapivin, Maurizio Marchese, Andrei Yadrantsau, and Yanchun Liang. Unsupervised key-phrases extraction from scientific papers using domain and linguistic knowledge. In *Proceedings of ICDIM*, pages 105–112, 2008.

[50] John Lafferty and Chengxiang Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of SIGIR*, pages 111–119, 2001.

[51] Victor Lavrenko and W. Bruce Croft. Relevance-based language models. In *Proceedings of SIGIR*, pages 120–127, 2001.

[52] Nicholas Lester, Alistair Moffat, and Justin Zobel. Fast on-line index construction by geometric partitioning. In *Proceedings of CIKM*, pages 776–783, 2005.

[53] Jinyang Li, Boon Thau, Loo Joseph, M. Hellerstein, and M. Frans Kaashoek. On the feasibility of peer-to-peer web indexing and search. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, 2003.

[54] Shuang Liu, Fang Liu, Clement Yu, and Weiyi Meng. An effective approach to document retrieval via utilizing WordNet and recognizing phrases. In *Proceedings of SIGIR*, pages 266–272, 2004.

[55] Toan Luu, Gleb Skobeltsyn, Fabius Klemm, Maroje Puh, Ivana Podnar Zarko, Martin Rajman, and Karl Aberer. AlvisP2P: scalable peer-to-peer text retrieval in a structured p2p network. In *Proceedings of VLDB Endowment*, 2008.

[56] Wenhui Ma, Wenbin Fang, Gang Wang, and Jing Liu. Concept index for document retrieval with peer-to-peer network. In *Proceedings of SNPD*, 2007.

[57] Rila Mandala, Takenobu Tokunaga, and Hozumi Tanaka. Combining multiple evidence from different types of thesaurus for query expansion. In *Proceedings of SIGIR*, pages 191–197, 1999.

[58] Christoph Mangold. A survey and classification of semantic search approaches. *Journal on Metadata Semantics and Ontology*, 2:23–34, 2007.

[59] Christopher Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[60] George Miller. *WordNet: An electronic Lexical Database*. MIT Press, 1998.

[61] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *TOIS*, 14:349–379, 1996.

[62] Alistair Moffat and Justin Zobel. Exploring the similarity space. *SIGIR Forum*, 32:18–34, 1998.

[63] Dan Moldovan and Rada Mihalcea. Using WordNet and lexical operators to improve internet searches. *IEEE Internet Computing*, 4:34–43, 2000.

[64] Roberto Navigli. Word sense disambiguation: A survey. *ACM Computing Surveys*, 41:1–69, 2009.

[65] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella: A P2P networking infrastructure based on RDF. In *Proceedings of WWW*, 2002.

[66] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st international conference on Scalable information systems (InfoScale)*, 2006.

[67] Jay Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of SIGIR*, pages 275–281, 1998.

[68] Martin Porter. An algorithm for suffix stripping. *Program*, 14:130–137, 1980.

[69] Yonggang Qiu and Hans-Peter Frei. Concept based query expansion. In *Proceedings of SIGIR*, pages 160–169, 1993.

[70] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of SIGCOMM*, 2001.

[71] John Risson and Tim Moors. Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks*, 50:3485–3521, 2006.

[72] Stephen Robertson. The probability ranking principle in IR. *Journal of documentation*, 33:294–304, 1977.

[73] Stephen Robertson. How Okapi came to TREC. In *Proceedings of TREC*, pages 287–299. The MIT Press, 2005.

[74] Stephen Robertson and Steve Walker. Okapi/Keenbow at TREC-8. In *Proceedings of TREC*, 1999.

[75] J. Rocchio. Relevance feedback in information retrieval. In *Salton G, Ed., The Smart Retrieval System– Experiments in Automatic Document Processing*, pages 313–323. Prentice-Hall Inc., 1971.

[76] Peter Mark Roget. *Roget's International Thesaurus.* Thomas Y. Crowell, 1946.

[77] Antony Rowstron and Peter Druschel. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of ACM SIGCOM*, 2001.

[78] Ian Ruthven and Mounia Lalmas. A survey on the use of relevance feedback for information access systems. *Knowledge Engineering Review*, 18:95–145, 2003.

[79] Gerard Salton, editor. *The SMART Retrieval System – Experiments in Automatic Document Processing.* Prentice Hall, 1971.

[80] Mark Sanderson. Retrieving with good sense. *Information Retrieval*, 2:49–69, 2000.

[81] Hinrich Schutze and Jan O. Pedersen. Information retrieval based on word senses. In *Proceedings of 4th Annual Symposium on Document Analysis and Information Retrieval*, 1995.

[82] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34:1–47, 2002.

[83] Gleb Skobeltsyn and Karl Aberer. Distributed cache table: efficient query-driven processing of multi-term queries in P2P networks. In *Proceedings of the international workshop on Information retrieval in peer-to-peer networks (P2PIR)*, 2006.

[84] John Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, 1984.

[85] Karen Spärck Jones, S. Walker, and Stephen E. Robertson. A probabilistic model of information retrieval: Development and comparative

experiments. *Information Processing and Management*, 36:779–808, 809–840, 2000.

[86] Kunwadee Spripanidkulchai, Bruce Maggs, and Hui Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceedings of IEEE INFOCOM*, pages 2166–2176, 2003.

[87] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM*, 2001.

[88] Christopher Stokoe, Michael P. Oakes, and John Tait. Word sense disambiguation in information retrieval revisited. In *Proceedings of SIGIR*, pages 159–166, 2003.

[89] Gilbert Strang, editor. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[90] Trevor Strohman and W. Bruce Croft. Efficient document retrieval in main memory. In *Proceedings of SIGIR*, pages 175–182, 2007.

[91] Chunqiang Tang and Sandhya Dwarkadas. Hybrid global-local indexing for effcient peer-to-peer information retrieval. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[92] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of ACM SIGCOMM*, pages 175–186, 2003.

[93] Peter Turney. Learning algorithms for keyphrase extraction. *Information Retrieval*, 2:303–336, 2000.

[94] Cornelis Joost van Rijsbergen. *Information Retrieval*. Butterworths, 1979.

[95] Ellen Voorhees. Query expansion using lexical-semantic relations. In *Proceedings of SIGIR*, pages 61–69, 1994.

[96] Ellen Voorhees. Overview of TREC 2006. In *Proceedings of TREC*, 2006.

[97] William Woods. Conceptual indexing: A better way to organize knowledge. Technical Report TR-97-61, Sun Microsystems Laboratories, USA, 1997.

[98] Ilya Zaihrayeu, Lei Sun, Fausto Giunchiglia, Wei Pan, Qi Ju, Mingmin Chi, and Xuanjing Huang. From web directories to ontologies: Natural language processing challenges. In *Proceedings of ISWC 2007*, 2007.

[99] Chengxiang Zhai. Fast statistical parsing of noun phrases for document indexing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 312–319, 1997.

[100] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of SIGIR*, pages 334–342, 2001.

[101] Jiangong Zhang and Torsten Suel. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2005.

[102] B. Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Computer Science Department, University of California, 2001.

[103] Yingwu Zhu and Yiming Hu. ESS: Efficient semantic search on Gnutella-like P2P system. Technical report, Department of ECECS, University of Cincinnati, 2004.

[104] Justin Zobel and Philip Dart. Finding approximate matches in large lexicons. *Software Practice and Experience*, 25:331–345, 1995.

[105] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38, 2006.

[106] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23:453–490, 1998.

Appendix A

Correctness and Completeness

In order to show that Algorithm 1 in Section 4.2 is sound and complete, we need to prove the following theorem:

Theorem A.0.1. *Let A' and B' be atomic concepts, and $\mathcal{T}_{WN}$ be a terminological knowledge base which can be represented as an acyclic graph, where nodes are atomic concepts and edges are subsumption axioms in the form $A' \sqsubseteq B'$. Then:*

$$\mathcal{T}_{WN} \models \sqcap A' \sqsubseteq \sqcup \sqcap B' \text{ iff there exists } \sqcap B' \text{ in } \sqcup \sqcap B', s.t., \mathcal{T}_{WN} \models \sqcap A' \sqsubseteq \sqcap B' \tag{A.1}$$

Note that, in Equation A.1, by $\sqcap A$ we denote conjunction ($\sqcap$) of atomic concepts (A) without negation and by $\sqcup \sqcap A$ we denote disjunctions ($\sqcup$) of $\sqcap A$.

Proof. It is known, that a subsumption problem with respect to an acyclic terminological knowledge base can be reduced to a subsumption problem with respect to the empty knowledge base [5]:

$$\mathcal{T}_{WN} \models D' \sqsubseteq E' \iff \models D \sqsubseteq E \tag{A.2}$$

where (complex) concepts D and E are obtained by replacing each occurrence of atomic concept A' in (complex) concepts D' and E' by the

conjunction $\sqcap A$ of all atomic concepts A from $\mathcal{T}_{WN}$ which are more general than or equivalent to A'.
Given A.2, we can rewrite Equation A.1 as follows:

$$\models \sqcap A \sqsubseteq \sqcup \sqcap B \text{ iff there exists } \sqcap B \text{ in } \sqcup \sqcap B, \text{s.t., } \models \sqcap A \sqsubseteq \sqcap B \tag{A.3}$$

□

Now, in order to prove Equation A.1, it is enough to prove Equation A.3. In the following we first prove the "if" direction of Equation A.3 and later we demonstrate the proof for the "only if" direction of Equation A.3.

If. Recall that disjunction ("$\sqcup$") is distributive over conjunction ("$\sqcap$"), i.e., if A_1, A_2, and A_3 are concepts than $A_1 \sqcup (A_2 \sqcap A_3) \equiv (A_1 \sqcup A_2) \sqcap (A_1 \sqcup A_3)$. By using the distributive property of disjunction we can convert concept $\sqcup \sqcap B$ from DNF into CNF (we use indexes i,j,k,l in order to enumerate atomic concepts):

$$\sqcup_i \sqcap_j B_{ij} \equiv \sqcap_k \sqcup_l C_{kl} \tag{A.4}$$

Notice, that concepts B_{ij} and C_{kl} in Equation A.4 satisfy the following property:

$$\begin{array}{l}\text{for all the possible combinations } B_1, \ldots, B_I \text{ of atomic concepts } B, \\ \text{where an atomic concept } B_i \text{ is taken from i-th conjunctive clause } \sqcap_j B_{ij} \\ \text{in } \sqcup_i \sqcap_j B_{ij}, \text{there exists disjunctive clause } \sqcup_l C_{kl} \text{ in } \sqcap_k \sqcup_l C_{kl}, \text{ s.t.,} \\ \sqcup_l C_{kl} \text{ is composed from all and only atomic concepts in } \{B_1, \ldots, B_I\}.\end{array} \tag{A.5}$$

Given A.4, subsumption $\models \sqcap A \sqsubseteq \sqcup \sqcap B$ in Equation A.3 can be rewritten as follows:

$$\models \sqcap A \sqsubseteq \sqcap \sqcup C \tag{A.6}$$

A concept can be subsumed by a conjunction of concepts if and only if it is subsumed by every concept in the conjunction:

$$\models \sqcap A \sqsubseteq \sqcap \sqcup C \text{ iff for all } \sqcup C \text{ in } \sqcap \sqcup C, \models \sqcap A \sqsubseteq \sqcup C \tag{A.7}$$

Recall that if A_1 and A_2 are concepts, then:

$$A_1 \sqsubseteq A_2 \text{ iff } A_1 \sqcap \neg A_2 \sqsubseteq \bot \tag{A.8}$$

$$\neg(A_1 \sqcup A_2) \equiv \neg A_1 \sqcap \neg A_2 \tag{A.9}$$

Given A.8 and A.9, subsumption $\models \sqcap A \sqsubseteq \sqcup C$ in Equation A.7 can be rewritten as follows:

$$\models \sqcap A \sqsubseteq \sqcup C \text{ iff } \models (\sqcap A) \sqcap (\sqcap \neg C) \sqsubseteq \bot \tag{A.10}$$

From A.10, it follows that (a) there exists a pair of atomic concepts A and C which have the same name; or (b) there exists an atomic concept $A \equiv \bot$; or (c) there exists an atomic concept $C \equiv \top$. From above, it follows that there exists a pair of atomic concepts A and C, such that, A is more specific than C.

$$\models \sqcap A \sqsubseteq \sqcup C \text{ iff there exists } A \text{ and there exists } C \text{ , s.t., } A \sqsubseteq C \tag{A.11}$$

Recall that if at least one concept A in conjunction $\sqcap A$ is subsumed by concept C, then the whole conjunction $\sqcap A$ is also subsumed by C. Taking it into account and using Equation A.11 we can prove that:

$$\models \sqcap A \sqsubseteq \sqcup C \text{ iff there exists } C \text{ in } \sqcup C, \text{s.t., } \models \sqcap A \sqsubseteq C \tag{A.12}$$

Given A.12, second part of Equation A.7 can be rewritten as follows:

$$\boxed{\text{for all } \sqcup C \text{ there exists } C \text{ in } \sqcup C, \text{s.t., } \models \sqcap A \sqsubseteq C} \tag{A.13}$$

Now, let us assume that the "if" direction of Equation A.3 doesn't hold, i.e., concept $\sqcap A$ is not subsumed by any concept $\sqcap B$:

$$\text{for all } \sqcap B \text{ in } \sqcup \sqcap B, \not\models \sqcap A \sqsubseteq \sqcap B \tag{A.14}$$

Recall that a concept can be subsumed by a conjunction of concepts if and only if it is subsumed by every concept in the conjunction:

$$\models \sqcap A \sqsubseteq \sqcap B \text{ iff for all } B \text{ in } \sqcap B, \models \sqcap A \sqsubseteq B \tag{A.15}$$

Given A.15, Equation A.14 can be rewritten as follows:

$$\text{for all } \sqcap B \text{ in } \sqcup \sqcap B \text{ there exists } B \text{ in } \sqcap B \text{ s.t., } \not\models \sqcap A \sqsubseteq B \tag{A.16}$$

Given Property A.5, Equation A.16 can be rewritten as follows:

$$\boxed{\text{there exists } \sqcup C \text{ s.t., for all } C \text{ in } \sqcup C \not\models \sqcap A \sqsubseteq C} \tag{A.17}$$

Equation A.17 is in contradiction with Equation A.13. Therefore, we discard the assumption made in Equation A.14, which means that the "if" direction of Equation A.3 holds. □

Only-if. The union of concepts is more general or equivalent to every concept in the union:

$$\text{for all } \sqcap B \text{ in } \sqcup \sqcap B, \mathcal{T}_{WN} \models \sqcap B \sqsubseteq \sqcup \sqcap B \tag{A.18}$$

Recall that the subsumption is the transitive relation, i.e.,

$$\text{if } \mathcal{T}_{WN} \models \sqcap A \sqsubseteq \sqcap B \text{ and } \mathcal{T}_{WN} \models \sqcap B \sqsubseteq \sqcup \sqcap B, \text{ then } \mathcal{T}_{WN} \models \sqcap A \sqsubseteq \sqcup \sqcap B \tag{A.19}$$

From A.19, we can see that the "only-if" direction of Equation A.3 holds. □

Equation A.3 and consequently Theorem A.0.1 are proved.